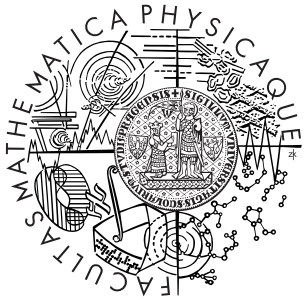
Univerzita Karlova v Praze
Matematicko-fyzikální fakulta

DIPLOMOVÁ PRÁCE



Matej Hrinčár

Konvoluční neuronové sítě a jejich využití při detekci objektů

Katedra teoretické informatiky a matematické logiky

Vedoucí diplomové práce: doc. RNDr. Iveta Mrázová, CSc.,

Studijní program: Informatika, Teoretická informatika

2012

Touto cestou by som rád poďakoval všetkým, ktorí ma podporovali pri písaní tejto práce. Obzvlášť by som rád poďakoval svojej vedúcej práce *doc. RNDr. Ivete Mrázovej, CSc.*, za kvalitné odborné vedenie, cenné rady a priebežnú kontrolu mojich výsledkov.

Prohlašuji, že jsem tuto diplomovou práci vypracoval(a) samostatně a výhradně s použitím citovaných pramenů, literatury a dalších odborných zdrojů. Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona v platném znění, zejména skutečnost, že Univerzita Karlova v Praze má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona.

V Praze dne 3. 12. 2012 Matej Hrinčár

Názov práce: Konvoluční neuronové sítě a jejich využití při detekci objektů
Autor: Matej Hrinčár
Katedra (ústav): Katedra teoretické informatiky a matematické logiky (32-KTIML)
Vedúci diplomovej práce: doc. RNDr. Iveta Mrázová, CSc.
E-mail vedúceho práce: Iveta.Mrazova@mff.cuni.cz

Abstrakt: V dnešnej dobe je moderné zatraktívňovať športové prenosy tzv. rožšírenou realitou, napríklad k hráčom hokejového zápasu zobraziť rôzne štatistiky. Aby sme to mohli urobiť, musíme hráčov najprv automaticky nájsť - detekovať. Touto náročnou úlohou sa zaoberá predložená práca. Išlo nám nielen o presnosť, ale i o rýchlosť, pretože by sme mali byť schopní detekcie v reálnom čase. Využili sme jeden z novších modelov neurónových sietí – *konvolučné siete*. Sú vhodné na spracovanie obrazových dát a ako vstup dostávajú obrázok bez akéhokoľvek predspracovania. Na základe podrobnej analýzy a urobených testov sme si ich vybrali pre implementáciu detektora hokejových hráčov v hokejovom zápase. V práci sme otestovali niekoľko rôznych architektúr týchto sietí, porovnali ich a vybrali tú z nich, ktorá je presná a rýchla. Otestovali sme i robustnosť siete na zašumených vzoroch. Nakoniec sme pre takto detekovaných hráčov použili farbu ich dresu a na jej základe ich pomocou algoritmu *K-means* zaradili do jedného z práve hrajúcich tímov.

Kľúčové slová: konvolučné siete, detekcia objektov, hokej, hráč

Title: Convolutional neural networks and their application in object detection
Author: Matej Hrinčár
Department: Department of Theoretical Computer Science and Mathematical Logic
Supervisor: doc. RNDr. Iveta Mrázová, CSc.
Supervisor's e-mail address: Iveta.Mrazova@mff.cuni.cz

Abstract: Nowadays, it has become popular to enhance live sport streams with an augmented reality like adding various statistics over the hockey players. To do so, players must be automatically detected first. This thesis deals with such a challenging task. Our aim is to deliver not only a sufficient accuracy but also a speed because we should be able to make the detection in real time. We use one of the newer model of neural network which is a convolutional network. This model is suitable for processing image data a can use input image without any preprocessing whatsoever. After our detailed analysis we choose this model as a detector for hockey players. We have tested several different architectures of the networks which we then compared and choose the one which is not only accurate but also fast enough. We have also tested the robustness of the network with noisy patterns. Finally we assigned detected players to their corresponding teams utilizing K-mean algorithm using the information about their jersey color.

Keywords: convolutional neural networks, object detection, hockey, players

Obsah

Úvod

Športové prenosy majú vysokú sledovanosť. Je však snahou ich zatraktívniť, aby divákovi priniesli ešte väčší zážitok zo športu. Grafická stránka prenosu je preto čím ďalej dôležitejšia a do popredia sa dostáva tzv. rozšírená realita, kde sa do živého obrazu dokresľujú rôzne informácie pomocou metód počítačovej grafiky. Aby sme to mohli urobiť, je potrebné vstupnému obrazu porozumieť. Ako príklad si vezmime hokejový zápas. Pri zábere na hraciu plochu by sme mohli k jednotlivým hokejistom vypisovať rôzne informácie. Najprv ale musíme hokejistov v obraze nájsť. A práve problémom detekovania hokejistov na hracej ploche sa zaoberá táto práca.

V tejto práci naštudujeme rôzne modely neurónových sietí a analyzujeme ich vlastnosti. Súčasťou tejto práce je implementácia vybraného modelu ako detektor hokejových hráčov na snímkach alebo videu z hokejového zápasu. Implementovaný bude pomocou konvolučných neurónových sietí, ktoré vychádzajú z poznatkov o fyziologických procesoch, ktoré nastavajú pri videní. Tieto siete nepotrebujú vopred určiť príznaky vzorov, ale sami si vo vzoroch nájdu, čo ich zaujíma. Do istej miery sú invariantné voči malým geometrickým transformáciám, ako je napríklad posun, zmena mierky, či rotácia. Práve pre tieto vlastnosti sme si ich vybrali ako model pre implementovanie programu pre detekciu hráčov.

Práca je rozdelená do niekoľkých kapitol. Na začiatku práce si v kapitole 1 popíšeme základný matematický model neurónu. Ten rozšírime do vrstevnatej neurónovej siete a popíšeme si jej učenie pomocou metódy spätného šírenia chyby - backpropagation.

V ďalšej kapitole 2 si predstavíme model s názvom Neocognitron. Ide o model inšpirovaný fyziologickými procesmi, ktoré nastávajú v mozgu pri videní a rozpoznávaní objektov. Tento model bol jedným z prvých, ktoré dokázali rozpoznávať ručne písané znaky a čísla. Vychádzajú z neho ďalšie modely neurónových sietí.

V kapitole 3 si predstavíme a popíšeme konvolučné neurónové siete, ktoré vychádzajú z modelu Neocognitron. Tieto siete sme si vybrali na implementáciu nášho riešenia. Povieme si z čoho sa skladajú a ako sa učia. Popíšeme ich štruktúru a algoritmus učenia.

V kapitole 4 si povieme niečo o predspracovaní obrazu. Konkrétne o geometrických transformáciách, odstraňovaní šumu a detekcii hrán.

Niekoľko architektúr konvolučných sietí navrhneme v kapitole 5. Tie otestujeme na rôznych množinách, vrátane zašumených vzorov a vyberieme model s najlepšou úspešnosťou a rýchlosťou vyhodnocovania. Správny pomer rýchlosti a úspešnosti je dôležitý pre rýchle rozpoznávanie hráčov, pretože ak by sme chceli výsledok použiť pre rozšírenú realitu, je potrebné aby bola detekcia možná v reálnom čase. V kapitole 6 podrobne analyzujeme vybranú neurónovú sieť.

Pred samotnou implementáciou v kapitole 7 navrhneme jednoduché rozšírenie detektora o zaradenie hráča do jedného z hrajúcich tímov.

V kapitole 8 vyberieme knižnicu, ktorá nám umožní použiť konvolučné neurónové siete. Nakoniec v kapitole 9 popíšeme implementáciu vytvorených programov na tvorbu vzorov, trénovanie a testovanie siete a programu slúžiacemu pre detekciu hráčov.

Práca ešte obsahuje dodatok B s užívateľskou dokumenáciou pre jednotlivé vytvorené programy.

Kapitola 1

Neurónové siete

V tejto diplomovej práci sa budeme podrobne zaoberať novým modelom neurónových sietí, konvolučnými neurónovými sieťami. Pre pochopenie týchto sietí si v tejto kapitole predstavíme matematický model neurónu, ktorý je stavebným prvkom každej umelej neurónovej siete. Ďalej tieto jednotlivé neuróny zapojíme do najznámejšieho modelu neurónových sietí - vrstevnatej neurónovej siete a popíšeme si proces učenia tejto siete pomocou algoritmu spätného šírenia chyby.

Umelé neurónové siete boli inšpirované biologickými neurónovými sieťami, ktoré sa nachádzajú v živých organizmoch. Ich základom sú bunky - biologické neuróny, navzájom prepojené do zložitých štruktúr. Biologický neurón [16] sa skladá z tela (soma), vstupných (dendridy) a výstupných (axony) výbežkov. Dendridy sú väčšinou krátke a bohato sa vetvia v blízkosti tela. Zabezpečujú privedenie vstupných signálov do tela neurónu. Axon, výstup neurónu, je iba jeden a môže byť veľmi dlhý. Na konci sa axon vetví a pripája na dendridy iných neurónov. Neuróny spolu komunikujú pomocou špecializovaných štruktúr - synapsií, ktoré sú buď inhibičné, alebo excitačné, teda buď signál potlačujú alebo zosilňujú. Neuróny spolu tvoria nervový systém, ktorý umožňuje organizmom reagovať na vonkajšie podnety.

Na základe pozorovania správania tejto významnej bunky bol navrhnutý matematický model - formálny neurón.

1.1 Formálny neurón

Formálny neurón je usporiadaná trojica $(\vec{w}, \vartheta, f)$ kde $\vec{w} = (w_1 \ldots w_n) \in \mathbb{R}^n$ je vektor váh, $\vartheta \in \mathbb{R}$ je prah a $f : \mathbb{R}^{n+1} \times \mathbb{R}^n \to \mathbb{R}$ je prenosová funkcia.

Na vstup neurónu je predložený vstupný vektor $\vec{x} = (x_1 \ldots x_n) \in \mathbb{R}^n$. Tieto vstupy sú zosilnené alebo potlačené váhami. Vstupy upravené váhami sa sčítajú a spolu s prahom dostaneme vnútorný potenciál neurónu, ktorý zrátame podľa (1.1)

$$\xi = \sum_{i}^{n} x_i w_i + \vartheta \tag{1.1}$$

Aby sme získali výstup neurónu, je potrebné na potenciál aplikovať prenosovú funkciu. Dostávame výstup neurónu:

$$y = f(\xi) \tag{1.2}$$

Celý vzorec:

$$y = f\left(\sum_{i}^{n} x_i w_i + \vartheta\right) \tag{1.3}$$

Výstup neurónu závisí na prenosovej funkcii. Používa sa viacero funkcií rôznych typov, my si popíšeme iba základné z nich. Najjednoduchšia je skoková prenosová funkcia (1.4), ktorá aktivuje výstup, ak je vnútorný potenciál neurónu kladný. Jej priebeh vidíme na obrázku 1.1.

$$f(\xi) = \begin{cases} 1 & \text{ak } \xi > 0 \\ 0 & \text{ak } \xi \leq 0 \end{cases} \tag{1.4}$$

Najčastejšie je však používaná sigmoidálna prenosová funkcia (1.5), kde λ určuje strmosť sigmoidy alebo prenosová funkcia v podobe hyperbolického tangentu (1.6). Je to najmä preto, lebo existujú spojité derivácie v každom bode. To sa neskôr využije pri učení neurónovej siete pomocou algoritmu spätného šírenia chyby. Na obrázkoch 1.2 a 1.3 vidíme grafické znázornenie ich priebehu.

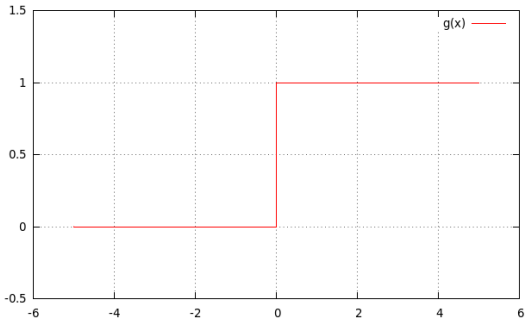
$$f(\xi) = \frac{1}{1 + e^{-\lambda \xi}} \tag{1.5}$$

$$f(\xi) = \frac{2}{1 + e^{-2x}} - 1 \tag{1.6}$$

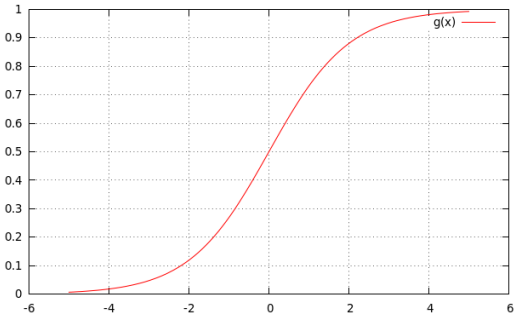
1.2 Neurónové siete

Vzájomným prepojením jednotlivých neurónov vznikne neurónová sieť. Prepojenie je realizované privedením výstupnej hodnoty neurónu na vstupy ďalších neurónov. Topológia siete je určená grafom predstavujúcim prepojenia medzi jednotlivými neurónmi. Vrcholmi sú jednotlivé neuróny a hrany odpovedajú prepojeniam.

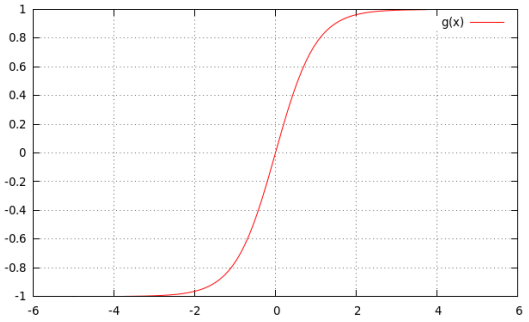
Matematicky je neurónová sieť usporiadaná 6-tica $M = (N, C, I, O, w, t)$, kde:



Obr. 1.1: Grafické znázornenie skokovej prenosovej funkcie.



Obr. 1.2: Grafické znázornenie sigmoidálnej prenosovej funkcie.



Obr. 1.3: Grafické znázornenie prenosovej funkcie hyperbolického tangentu.

- N je konečná a neprázdna množina neurónov,
- $C \subseteq N \times N$ je neprázdna množina orientovaných spojov medzi neurónmi,
- $I \subseteq N$ je neprázdna množina vstupných neurónov,
- $O \subseteq N$ je neprázdna množina výstupných neurónov,
- $w : C \to \mathbb{R}$ je váhová funkcia,
- $t : N \to \mathbb{R}$ je prahová funkcia.

Ak je graf určený topológiou siete acyklický a neuróny sa dajú usporiadať do vrstiev, kde spoje vedú iba medzi susednými vrstvami, hovoríme o *vrstevnatej sieti*. Vstupné neuróny I, ktoré tvoria vstupnú vrstvu, dostávajú na vstup každý jednu zložku vstupu a iba prenášajú vstupy ďalej do siete. Výstupné neuróny O tvoria výstupnú vrstvu a ostatné neuróny tvoria skryté vrstvy.

Výpočet siete prebieha tak, že na vstup siete predložíme požadovaný vstupný vektor. Ten je predložený na vstupy neurónov v prvej skrytej vrstve. Tie spočítajú svoj výstup a ten je predložený vrstve ďalšej. Pokračujeme až k výstupnej vrstve.

1.3 Učenie neurónovej siete

Na učenie vrstevnatej siete sa používa napríklad *algoritmus spätného šírenia* [14]. Je to gradientná metóda učenia s učiteľom. Porovnáva skutočný výstup oproti očakávanému a adaptuje váhy a prahy jednotlivých neurónov proti gradientu chybovej funkcie smerom od výstupnej vrstvy k vstupnej. Výhodou takto naučených sieti sú väčšinou dobré generalizačné vlastnosti, nevýhodou je možnosť uviaznutia v lokálnom minime a pomalá rýchlosť učenia.

Množinu vstupných vzorov si rozdeľme na množinu trénovacích, testovacích a prípadne validačných vzorov. Trénovacie vzory sa použijú pri učení siete.

Pre zistenie chyby siete si musíme najprv definovať chybovú funkciu [14]. Cieľom učenia je ju na trénovacej (testovacej) množine vzorov minimalizovať.

$$E = \frac{1}{2} \sum_{p} \sum_{j} (y_{j,p} - d_{j,p})^2 \tag{1.7}$$

Kde p je množina indexov vstupných vzorov, j sú indexy výstupných neurónov, y je skutočný výstup a d je očakávaná hodnota na výstupe.

Algoritmus používa na minimalizovanie chyby adaptáciu váh a to podľa [13]:

$$w_{ij}(t+1) = w_{ij}(t) + \alpha \Delta_E w_{ij}(t) + \alpha_m (w_{ij}(t) - w_{ij}(t-1)) \tag{1.8}$$

kde w_{ij} je váha vedúca z neurónu i do neurónu j, $\Delta_E w_{ij}(t)$ je prírastok váhy prispievajúci k minimalizácii E, α je parameter učenia, a α_m je moment učenia.

Pre výstupnú vrstvu spočítame prírastok ako:

$$\begin{aligned}\Delta_E w_{ij} &= -\frac{\partial E}{\partial w_{ij}} = -\frac{\partial E}{\partial \xi_j}\frac{\partial \xi_j}{\partial w_{ij}} = -\frac{\partial E}{\partial \xi_j}\frac{\partial}{\partial w_{ij}}\sum_k w_{kj}y_k = -\frac{\partial E}{\partial \xi_j}y_i = \\ &= -\frac{\partial E}{\partial y_j}\frac{\partial y_j}{\partial \xi_j}y_i = -(y_j - d_j)f'(\xi_j)y_i = \delta_j y_i \end{aligned} \tag{1.9}$$

Kde pre sigmoidálnu prenosovú funkciu dostávame:

$$f'(\xi_j) = y_j(1 - y_j)\lambda \tag{1.10}$$

Pre skryté vrstvy:

$$\begin{aligned}\Delta_E w_{ij} &= -\frac{\partial E}{\partial \xi_j}\frac{\partial \xi_j}{\partial w_{ij}} = -\sum_k \frac{\partial E}{\partial \xi_k}\frac{\partial \xi_k}{\partial y_j}\frac{\partial y_j}{\partial \xi_j}y_i = \\ &= -\Big(\sum_k \frac{\partial E}{\partial \xi_k}\frac{\partial}{\partial y_j}\sum_j w_{jk}y_j\Big)\frac{\partial y_j}{\partial \xi_j}y_i = \\ &= -\Big(\sum_k \frac{\partial E}{\partial \xi_k}\Big)\frac{\partial y_j}{\partial \xi_j}y_i = \\ &= \Big(\sum_k \delta_k w_{jk}\Big)f'(\xi_j)y_i = \delta_j y_i \end{aligned} \tag{1.11}$$

Celý proces učenia môžeme zapísať pomocou algoritmu 1.1.

Klasické vrstevnaté neurónové siete sa používajú v mnohých úlohách, či už ide o aproximáciu funkcii, či napríklad o rozpoznávanie obrazu. Ich výhodou je jednoduchý algoritmus učenia, nevýhodou je ich pomalosť. Na spracovanie veľkých obrazových dát sa tieto siete príliš nehodia, pretože nerešpektujú informácie z lokálneho okolia bodu. I malá zmena alebo posun obrazového vstupu spôsobí aktiváciu úplne iných neurónov. Preto sa pre učenie obrazových dát musí zredukovať veľkosť vstupu. Najčastejšie obraz transformuje do vstupného vektoru, ktorý popisuje určité vlastnosti obrazu, ako je napríklad veľkosť, farba, vzájomná poloha významných bodov atď. Jednotlivé príznaky by mali byť, v závislosti na úlohe, napr. invariantné voči posunu, rotácii a pod. Tieto vlastnosti i transformačnú funkciu z obrazu na danú zložku vstupného vektoru je nutné určiť a zostrojiť ručne. To však vyžaduje ďalšie znalosti.

Mnoho použitých klasických neurónových sietí obsahovalo iba málo skrytých vrstiev (1–3). Problémy nastávajú, keď je sieť veľmi hlboká, t.j. má veľa skrytých vrstiev.

Algorithm 1.1 Algoritmus učenia vrstevnatej neurónovej siete

Inicializácia parametrov siete (váhy, prahy) zväčša náhodne
$t \leftarrow 1$
repeat
for all trénovací vzor a jeho požadovaný výstup: $(\vec{x}, \vec{d})$ **do**
1. Vypočítame skutočný výstup siete:
Postupujeme od vstupnej vrstvy k výstupnej. Ako vstup prvej vrstvy použijeme trénovací vzor $\vec{x}$.
Aktivita neurónu v jednej vrstve je:
$y_j = f(\xi) = \frac{1}{1+e^{-\lambda \xi_j}}$, kde $\xi_j = \sum_i y_i w_{ij}$
Vypočítané aktivity tvoria vstup dalšej vrstvy.
2. Aktualizácia váh:
Postupujeme od výstupnej vrstvy k vstupnej. Adaptujeme podľa vzorca:
$w_{ij}(t+1) = w_{ij}(t) + \alpha \delta_j y_i + \alpha_m (w_{ij}(t) - w_{ij}(t-1))$
if aktualizujeme výstupnú vrstvu **then**
$\delta_j = (d_j - y_j) y_j (1 - y_j) \lambda$
else
$\delta_j = \lambda y_j (1 - y_j) \sum_k \delta_k w_{jk}$
end if
{ Kde:
$w_{ij}(t)$ - váha neurónu i do neurónu j v čase t.
α, α_m - parameter učenia a moment učenia.
ξ_j - potenciál neurónu j.
δ_j - chyba na neuróne j.
k - index neurónov z vrstvy za neurónom j.
λ - strmosť prenosovej funkcie.
}
$t \leftarrow t + 1$
end for
Spočítame chybu siete podľa:
$E = \frac{1}{2} \sum_p \sum_j (y_{j,p} - d_{j,p})^2$
kde p je množina indexov trénovacích vzorov, j sú indexy výstupných neurónov, y je skutočný výstup siete a d je očakávaný výstup siete.
until nebude splnené ukončovacie kritérium
{ukončovacie kritérium je napríklad počet trénovacích iterácii či veľkosť chyby siete.}

Pri učení takýchto sietí klasickým učiacim algoritmom sú výsledky horšie, ako pri sietiach s jednou alebo dvoma skrytými vrstvami (plytké siete). Pri použití algoritmu, ktorý začína s náhodne zvolenými váhami sa stáva, že učenie skončí v lokálnom minime.

Prečo ale používať hlboké neurónové siete? Dôvod pre ich masívne nasadenie v praxi spočíva v tom, že neocognitronové a konvolučné siete napodobňujú vizuálny systém živých organizmov. Rozpoznávanie sa rozkladá na viac úrovní. Každá úroveň reprezentuje istú mieru abstrakcie vstupného vzoru. Od úrovní, ktoré detekujú jednoduché lokálne príznaky, až po úrovne, ktoré v predkladaných vzoroch rozpoznajú celé objekty. Tento princíp môžeme použiť i v klasických dopredných vrstevnatých sietiach. Siete tohto typu budú mať mnoho skrytých vrstiev. To dovoľuje hierarchicky rozložiť problém na viac častí a tým urýchliť proces učenia. Jedna z metód pre učenie hlbokých sietí bola navrhnutá až v roku 2006 v [9]. Pri učení bolo použité predučenie bez učiteľa, potom bola sieť doučená s učiteľom. Navrhnutú hlbokú sieť (tzv. *Deep belief network*) tvoria vrstvy z Obmedzených Boltzmanových strojov *(Restricted Boltzmann machines, RBM)*. Neskôr navrhnutá varianta hlbokej siete sa skladá sa z vrstiev autoenkóderov. Obe tieto hlboké siete uspeli pri učení, práve kvôli predtrénovaniu bez učiteľa.

Ďalšiu z mála výnimiek tvorili až donedávna práve konvolučné siete, ktoré majú veľa vrstiev a dajú sa efektívne a dobre naučiť pomocou algoritmu spätného šírenia. V ďalších kapitolách si predstavíme tieto konvolučné neurónové siete, ktoré na vstupe majú priamo obraz a nemajú problém s menšími transformáciami vstupných obrázkov.

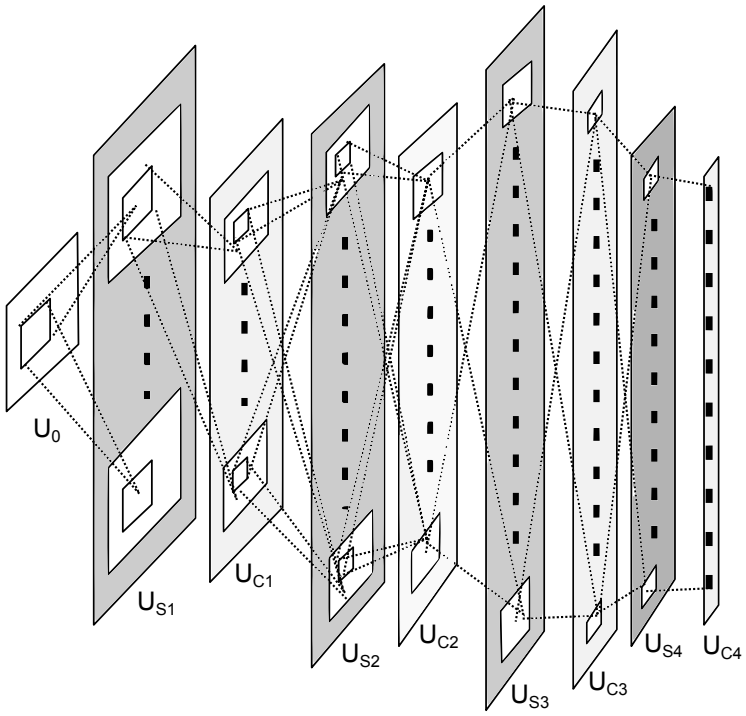
Kapitola 2

Neocognitron

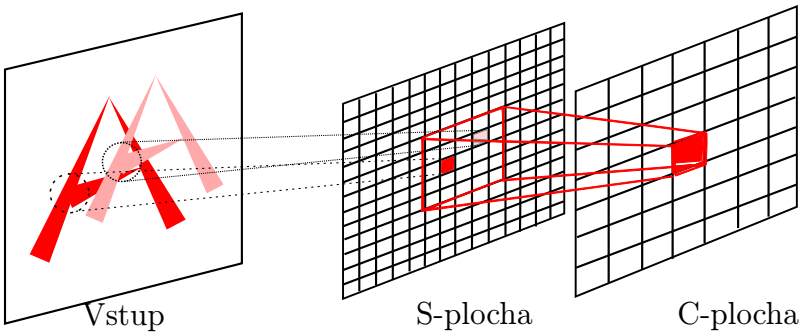
V tejto kapitole si popíšeme jednu z prvých neurónových sietí, ktoré na vstupe predpokladajú priamo obrazové dáta. Touto sieťou je sieť *Neocognitron*. *Neocognitron* je vrstevnatá neurónová sieť navrhnutá K. Fukushimom [4, 8]. Jej robustnosť pri rozpoznávaní vizuálnych vzorov bola ukázaná na príklade rozpoznávania ručne písaných znakov a číslic [6]. Tento model neurónovej siete napodobňuje vizuálny systém živých organizmov. Sieť má hierarchickú a viacvrstvovú architektúru podobne, ako sa predpokladá, že to je v časti mozgovej kôry, ktorá sa stará o videnie [10]. Teória predpokladá nasledujúcu štruktúru zrakovej kôry: jednoduché bunky → komplexné bunky → nízkoúrovňové hyperkomplexné bunky → vysokoúrovňové hyperkomplexné vrstvy. Ďalej predpokladá, že spojenia medzi jednoduchými a komplexnými bunkami sú podobné ako medzi nízko a vysoko úrovňovými bunkami. Architektúra neocognitronu bola navrhnutá na týchto fyziologických princípoch [6]. V tejto práci síce sieť neocognitron nie je implementovaná alebo použitá na detekciu objektov, ale jej princíp bol základom konvolučných sietí, ktoré v práci používame. Preto tento model aspoň stručne popíšeme.

Typickú neocognitronovú sieť môžeme vidieť na obrázku 2.1. Sieť je založená na hierarchickom skladaní a detekcii príznakov. Základom neocognitronu sú striedajúce sa vrstvy *S-buniek* a *C-buniek*. Prvá vrstva U_0 zodpovedá sietnici živých organizmov. Obsahuje *receptorové bunky*, ktoré zachytávajú vstup a preposielajú ho ďalším vrstvám. Každá ďalšia úroveň U_n obsahuje *S-vrstvu* U_{Sn} skladajúcu sa z niekoľkých *S-plôch*, *C-vrstvu* U_{Cn} skladajúcu sa z rovnakého počtu *C-plôch* ako je vo *S-vrstve* a nakoniec jednej *V-vrstvy* U_{Vn}. Plocha je matica buniek rovnakého typu. Posledná *C-vrstva* obsahuje toľko plôch, skladajúcich sa z jednej bunky, koľko kategórii sme schopní detekovať. Výstupná hodnota priamo udáva mieru, do akej kategórie patrí vstupný vzor. Počet plôch na jednotlivých vrstvách zodpovedá počtu príznakov, ktoré bude úroveň schopná detekovať [6].

S-bunka sa stará o detekciu príznakov. Je napojená na niekoľko C-plôch predchádzajúcej C (resp. vstupnej) vrstvy. Jej vstupom sú iba výstupy tých buniek, ktoré spadajú do jej recepčného poľa (a priemerná aktivita získaná z V-vrstvy). Veľkosť



Obr. 2.1: Typická neocognitronová sieť. Vstupný obrázok U_0 je privedený na prvú S vrstvu skladajúcu sa z S-plôch, ktorých S-bunky spracovávajú dáta vstupného obrázku z ich recepčného poľa. S vrstva je napojená na odpovedajúcu C vrstvu, tieto dvojice sa striedajú až k poslednej vrstve C plôch o rozmeroch 1 x 1, ktoré značia rozpoznanú triedu.



Obr. 2.2: Zaistenie invariancie voči posunutiu. Posun príznaku v S-ploche nezmení výstup bunky v C-ploche.

recepčného poľa sa zvolí pri vytváraní siete. Každá S-bunka na jednej S-ploche detekuje rovnaký príznak. Bunka je aktívna, ak sa v jej recepčnom poli nachádza vzor, na ktorý má bunka detekovať. Pomocou parametrov sa dá ovplyvniť, do akej miery môže byť vstup deformovaný, aby ho bunka mohla ešte detekovať. Presný vzorec výpočtu hodnoty S-bunky nájdeme v [5]. Príznak, ktorý bude bunka detekovať, sa určí v procese učenia. Na nižších vrstvách tieto bunky detekuju jednoduché príznaky, ako sú hrany, rohy a pod., na vyšších vrstvách detekujú zložitejšie štruktúry vstupného obrázku.

C-plocha je napojená svojim recepčným poľom na výstupy S-plochy. Skladá sa z C-buniek, ktoré v rovnakej ploche počítajú tú istú funkciu. Jej výstup závisí na počte aktívnych S-buniek na jej vstupe, ale stačí i jedna aktívna bunka na aktiváciu jej výstupu. Tým, že sa recepčné polia prekrývajú, prejaví sa jedna aktívna S-bunka na výstupe viacerých C-buniek. C-plocha tak zaisťuje istú mieru invariancie pri posunutí príznakov. Ak je nejaký príznak detekovaný S-bunkou mierne posunutý, ale spadá do recepčného poľa C-bunky, bude táto bunka aktivovaná nezávisle na posune príznaku. To je vidno na obrázku 2.2. C-plochy sú menšie, ako odpovedajúce S-plochy, preto klesá ich veľkosť smerom k výstupu siete. Plochy v poslednej vrstve U_{Cn} obsahujú už iba po jednej bunke. Tieto bunky zodpovedajú rozpoznávaným kategóriám a ich výstup priamo udáva mieru pravdepodobnosti danej kategórie.

Pôvodne bola sieť učená bez učiteľa, existujú ale i varianty učenia s učiteľom. Pri učení s učiteľom rozhodne učiteľ, aké príznaky a na akých úrovniach budú detekované. S vybraným príznakom určí i tzv. *seed-bunku* na ploche, ktorá bude tento príznak detekovať. Po predložení trénovacieho vzoru (príznaku) upraví učiteľ váhy tak, aby daná bunka príznak detekovala. Na prvú vrstvu sú zväčša predkladané príznaky v podobe rôzne natočených čiar a v ďalších vrstvách ich kombinácie. Učenie prebieha postupne od vstupnej vrstvy k výstupnej tak, že ďalšia vrstva môže byť trénovaná až po dokončení učenia predchádzajúcich vrstiev [5]. Pri učení bez učiteľa sa používa princíp *"víťaz berie všetko"*. Po predložení vzoru sa za *seed-bunku* vyberie bunka, ktorá ma na vzor najväčšiu odozvu. Vďaka zdieľaným váham v jednej ploche tak budú zosilnené vstupy všetkých buniek v danej ploche. Oproti sieťam, ktoré sa

učia pomocou algoritmu spätného šírenia, má neocognitron výhodu, že učený vzor stačí sieti predložiť omnoho menejkrát [7].

Neocognitron sa ukázal ako robustný nástroj na rozpoznávanie vzorov, odolávajúci zmenám na vstupe ako posun, rotácia a šum. Pri rozpoznávaní ručne písaných číslic dosiahol úspešnosť nad 98% [6]. Neocognitronová sieť je vhodná na rozpoznávanie jednoduchých a málo rozmerných obrázkov. V tejto práci ju implementovať ani priamo používať nebudeme. Jej základné princípy boli vo väčšej alebo menšej miere použité v ďalších systémoch na rozpoznávanie zložitejších objektov, napríklad i v konvolučných neurónových sietiach.

Kapitola 3

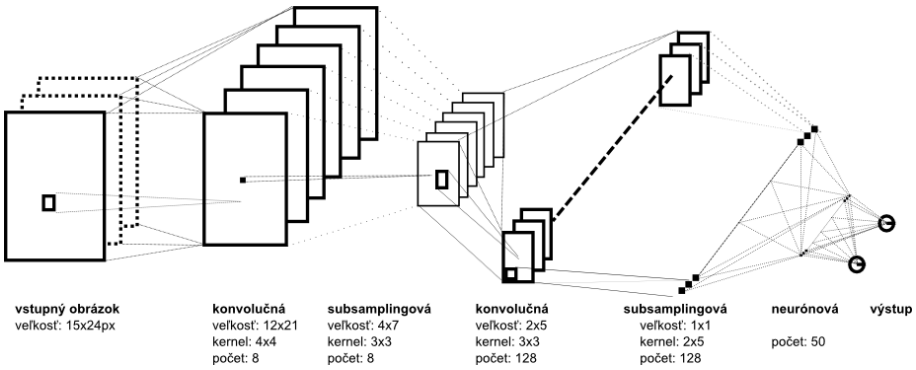
Konvolučné neurónové siete

Konvolučné neurónové siete sú špeciálnym druhom neurónových sietí, ktoré vychádzajú z princípov neocognitronovej siete. Boli navrhnuté pre rozpoznávanie dvojrozmerných obrazových vzorov priamo z pixelov obrázkov s minimálnym predspracovaním [11]. Pri spracovaní vizuálnych dát (ako je napríklad rozpoznávanie písaných číslic, písmen, tvárí) je vhodné, aby odozva siete nebola závislá na posune, mierke alebo deformovaní vstupných dát. Konvolučné siete to do určitého stupňa spĺňajú vďaka svojim vlastnostiam. Je to najmä získavanie príznakov z recepčných polí vstupných neurónov, zdieľanie váh a priestorové prevzorkovanie. Vďaka neurónom, spracúvajúcich iba vstupy zo svojich recepčných polí, je možné z obrázku získať základné príznaky, ako sú hrany, rohy a pod. Tieto príznaky sú potom v ďalších vrstvách navzájom kombinované, a tak získame príznaky na vyššej úrovni. Odpadá preto nutnosť zložitého predspracovania a získavania príznakov [12].

Obrázok 3.1 zobrazuje typickú konvolučnú sieť. Vstupný obrázok je spracovávaný vstupnou konvolučnou vrstvou. Jej výstup je privedený na prvú subsamplingovú vrstvu. Jej výstup je znova spracovaný ďalšou vrstvou. Striedajú sa tak konvolučné a subsampligové vrstvy. Nakoniec je výstup poslednej subsamplingovej vrstvy transformovaný do jednorozmerného vektoru a poslaný do plne prepojenej vrstevnatej neurónovej siete.

Významným prvkom konvolučných sietí sú tzv. *konvolučné vrstvy*. Každý neurón v tejto vrstve je prepojený s nejakým malým okolím (recepčné pole) k spracovávanemu obrázku na zodpovedajúcich súradniciach v (niekoľkých) predchádzajúcich vrstvách.

Podľa [12] sú neuróny v tejto vrstve organizované do rovín, v ktorých všetky neuróny zdieľajú tú istú množinu váh. Množina výstupov neurónov v tejto rovine sa nazýva *príznaková mapa*. Všetky neuróny v danej príznakovej mape teda počítajú rovnakú operáciu, len na rôznych častiach obrázku. Zdieľanie váh v jednej mape umožňuje obmedziť celkový počet výpočtov pri učení, a tak celý proces značne urýchliť. Rovnako sa tým obmedzuje pamäťová náročnosť. Jedna konvolučná vrstva



Obr. 3.1: Typická konvolučná sieť, vstupný obrázok je privedený na prvú konvolučnú vrstvu, skladajúcu sa z konvolučných príznakových máp, ktoré zdieľajú svoje parametre. Ich výstup je ďalej privedený na odpovedajúcu subsamplingovú vrstvu, ktorá výstup konvolučnej siete zmenší. Tieto dvojice vrstiev sa striedajú. Nakoniec je pripojená klasická vrstevnatá neurónová sieť.

je tvorená viacerými príznakovými mapami, a tak získavame niekoľko príznakov nad každou pozíciou vstupného obrázku. Základnou funkciou, ktorú neuróny v týchto vrstvách počítajú, je konvolúcia.

Konvolúcia je operátor, ktorý pracuje s dvoma funkciami.

Pri spracovaní obrazu sa používa diskrétny model konvolúcie. Vzorec na výpočet v bode (x, y) je nasledujúci:

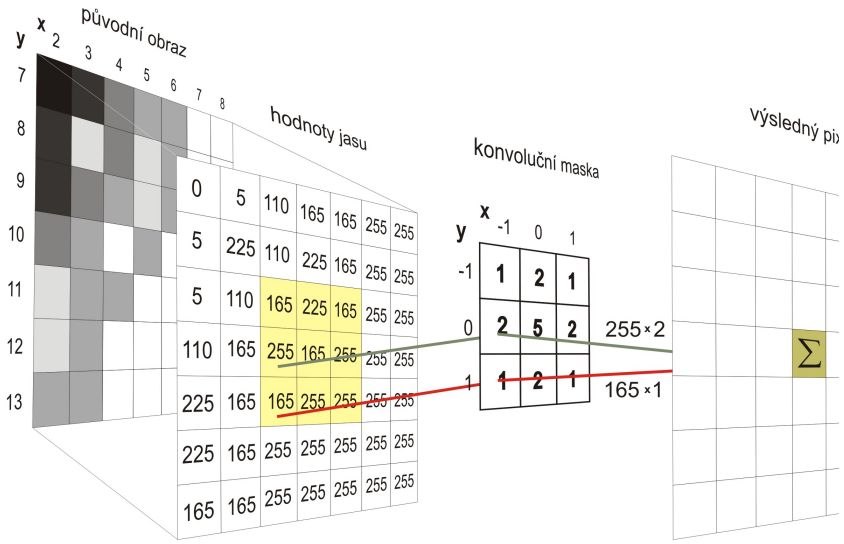
$$\mathbf{f} * \mathbf{k} = f(x, y) * k(x, y) = \sum_{(m,n)\in O} k(x - m, y - n) f(m, n) \tag{3.1}$$

$f(x, y)$ vracia hodnotu pixlu na daných súradniciach vo vstupnom obrázku $\mathbf{f}$, $k(i, j)$ vracia hodnotu na daných súradniciach tzv. konvolučné jadro $\mathbf{k}$. O je okolie bodu o veľkosti $m \times n$. Rovnakú veľkosť má i konvolučné jadro. V diskrétnom prípade ide o okienkový operátor definovaný maticou. Grafické znázornenie môžeme vidieť na obrázku 3.2.

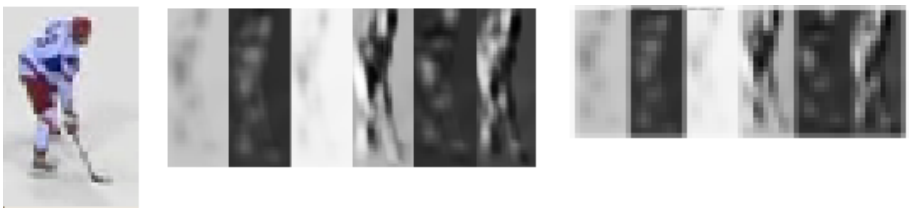
Každá *konvolučná vrstva* môže kombinovať výstupy viacerých príznakových máp z predchádzajúcej vrstvy. Podľa [2] dostávame výstup j-tej príznakovej mapy v l-tej konvolučnej vrstve:

$$\mathbf{x}_j^l = f\Big(\sum_{i\in M_j} \mathbf{x}_i^{l-1} * \mathbf{k}_{ij}^l + \vartheta_j^l \Big) \tag{3.2}$$

Kde M_j reprezentuje výber zo vstupných máp, teda určuje prepojenie, ktorá mapa bude prepojená s ktorou mapou v predchádzajúcej vrstve. $\mathbf{x}_i^{l-1}$ je i-tá príznaková mapa v $l-1$ (predchádzajúcej) vrstve, $\mathbf{k}_{ij}^l$ je konvolučné jadro j-tej príznakovej mapy v l-tej vrstve, pri prepojení na i-tú príznakovú mapu v predchádzajúcej vrstve.



Obr. 3.2: Grafické znázornenie diskrétnej konvolúcie. Zdroj: *wikipedia*.



Obr. 3.3: Vľavo je vstupný obrázok. Nasleduje výstup konvolučnej vrstvy so šiestimi príznakovými mapami a výstup napojenej subsamplingovej vrstvy. Vidíme, že každá príznaková mapa v konvolučnej vrstve detekuje iný príznak. Subsamplingová mapa zmenšuje výsledné rozlíšenie. V tomto prípade trikrát.

Každá príznaková mapa má vlastný aditívny parameter (prah) ϑ. Prenosová funkcia je f.

Medzi jednotlivé konvolučné vrstvy sa z dôvodu zníženia výpočtového času, ale najmä kvôli obmedzeniu závislosti na presnej polohe príznakov, vkladajú tzv. *subsamplingové* vrstvy. Každá mapa v tejto vrstve zodpovedá jednej (konvolučnej) mape v predchádzajúcej vrstve. Často táto vrstva vypočítava/priemeruje hodnoty v neprekrývajúcich sa blokoch veľkosti $N \times N$ vstupného obrázku. Teda veľkosť výstupu tejto subsamplingovej vrstvy bude N-krát menšia. Formálne by sa to dalo podľa [2] zapísať ako:

$$\mathbf{x}_j^l = f(\beta_j^l \text{down}(\mathbf{x}_j^{l-1}) + \vartheta_j^l) \tag{3.3}$$

Kde down$(\cdot)$ je prevzorkovacia funkcia. Každá mapa v tejto vrstve má vlastný multiplikatívný β a aditívny ϑ parameter.

Pre predstavu môžeme na obrázku 3.3 vidieť vstupný obrázok, výstup z prvej konvolučnej vrstvy obsahujúci 6 príznakových máp a výstup z nasledujúcej subsamplingovej vrstvy.

Výstup z príznakovej mapy poslednej konvolučnej alebo subsamplingovej vrstvy je transformovaný do jednorozmerného vektoru a privedený ako vstup do vrstevnatej neurónovej siete, ktorá spočíta konečný výstup.

3.1 Učenie konvolučnej siete

Konvolučnú sieť učíme metódou spätného šírenia. Základný princíp tohto učenia sme vysvetlili v kapitole 1.3. Tento algoritmus aplikujeme na konvolučnú sieť. Konvolučná sieť sa skladá zo striedajúcich sa konvolučných a subsamplingových vrstiev a môže byť zakončená klasickou plne prepojenou vrstevnatou sieťou. Pre koncovú časť siete môžeme použiť priamo algoritmus popísaný vyššie. Ostatné dva druhy vrstiev si popíšeme detailnejšie.

Každá konvolučná vrstva *(l-tá)* je nasledovaná subsamplingovou vrstvou *(l+1)*. Podľa [2] pre spočítanie chyby v jednej vrstve potrebujeme spočítať chyby cez všetky jednotky v nasledujúcej vrstve. Jeden pixel v nasledujúcej subsamplingovej vrstve zodpovedá bloku pixlov v konvolučnej vrstve. Pre účinné vypočítanie chýb l-tej konvolučnej vrstvy môžeme zobrať mapu chýb $l+1$ vrstvy a zväčšiť ju na rozmer konvolučnej vrstvy. Dostávame chybu j-tej mapy v l-tej vrstve:

$$\delta_j^l = \beta_j^{l+1}\Big(f'(\xi_j^l) \circ \mathrm{up}(\delta_j^{l+1})\Big) \tag{3.4}$$

kde funkcia up$(\cdot)$ (3.5) je zväčšovacia operácia, ktorá jednoducho ukladá každý pixel horizontálne a vertikálne n-krát, kde n je veľkosť recepčného poľa napojenej subsamplingovej vrstvy, ξ_j^l je potenciál neurónu v j-tej mape a v l-tej vrstve a nakoniec $f'(\cdot)$ je derivácia prenosovej funkcie.

$$\mathrm{up}(\mathbf{x}) = \mathbf{x} \otimes 1_{n\times n} \tag{3.5}$$

Teraz môžeme spočítať gradient prahu sčítaním všetkých položiek δ_j^l:

$$\frac{\partial E}{\partial \vartheta_j} = \sum_{u,v} (\delta_j^l)_{uv} \tag{3.6}$$

A nakoniec spočítame gradient váh podobne ako v klasickom algoritme spätného šírenia chyby s tým, že ta istá váha je zdieľaná s viacerými spojeniami.

$$\frac{\partial E}{\partial \mathbf{k}_{ij}^l} = \sum_{u,v} (\delta_j^l)_{uv} (\mathbf{p}_i^{l-1})_{uv} \tag{3.7}$$

Kde $(\mathbf{p}_i^{l-1})_{uv}$ je oblasť v $\mathbf{x}_i^{l-1}$, ktorá bola násobená s $\mathbf{k}_{ij}^l$ počas konvolúcie na vypočítanie bodu v konvolučnej mape $\mathbf{x}_i^l$ na súradniciach (u, v).

Pri adaptácii subsamplingových vrstiev je problémom vypočítať prenos chyby. Akonáhle ho máme, môžeme adaptovať parametre β a ϑ. Ak nasledujúca vrstva je plne prepojená vrstva, chyby vypočítame klasickým backpropagation algoritmom. Pri počítaní gradientu konvolučného jadra potrebujeme zistiť, ktoré oblasti sú napojené na daný pixel v ďalšej vrstve. Znova podľa [2], váhy vynásobené spojeniami medzi vstupnými a výstupnými pixelmi sú presne váhy (otočeného) konvolučného jadra. To nám dáva efektívnu implementáciu za použitia konvolúcie:

$$\delta_j^l = f'(\xi_j^l) \circ (\delta_j^{l+1} * rot180(\mathbf{k}_j^{l+1})) \tag{3.8}$$

Aditívny parameter ϑ je suma cez všetky elementy:

$$\frac{\partial E}{\partial \vartheta_j} = \sum_{u,v} (\delta_j^l)_{uv} \tag{3.9}$$

Multiplikatívny parameter β zahŕňa originálnu prevzorkovaciu funkciu vypočítanú počas prechodu vzoru sieťou. Dostávame gradient pre β:

$$\frac{\partial E}{\partial \beta_j} = \sum_{u,v} (\delta_j^l \circ \mathrm{down}(\mathbf{x}_j^{l-1}))_{uv} \tag{3.10}$$

Podobne, ako u klasických vrstevnatých neurónových sietí, môžeme konvolučné siete učiť pomocou algoritmu spätného šírenia chyby (backpropagation). Vďaka zdieľaným váham je počet parametrov, ktoré treba naučiť, menší, a tak je doba učenia rýchlejšia. Ďalšou výhodou konvolučných sietí je, ako sme už spomenuli, že pracujú priamo s obrazom. Nemusíme im obraz predspracovávať do menej rozmerných vektorov. Rovnako im po naučení nevadia mierne posuny a geometrické transformácie vzorov. Oproti neocognitronovej sieti vynikajú tým, že sú schopné spracovať väčšie vstupné obrázky a na ich učenie môžeme použiť metódu backpropagation, na rozdiel od ručného vyberania príznakov pre jednotlivé vrstvy. Vďaka týmto vlastnostiam sú konvolučné neurónové siete vhodnejšie pre spracovanie reálnych obrazových dát, a preto ich použijeme na riešenie našej úlohy.

Kapitola 4

Predspracovanie obrazu

Pri spracovávaní obrazových dát sa stretávame s rôznymi problémami, ktoré sa vyskytujú na reálnych obrázkoch. Problémy robí najmä šum, ale i rôzne deformácie obrazu spôsobené napr. optickou sústavou, cez ktorú bol obraz získaný. Ďalej je v niektorých úlohách vhodné zvýrazniť určité časti obrazu pre ďalšie spracovanie, či už pre človeka, alebo pre program. Ide napríklad o zvýraznenie, či nájdenie hrán, rohov, významných bodov a podobne. Pri spracovaní videa môže byť problémom zobrazovanie pomocou prekladania snímok, ktoré je vhodné pred ich spracovaním vhodne odstrániť. V prípade klasických neurónových sietí, ktoré by spracovávali obrazové dáta, by sme museli použiť nejakú z techník predspracovania obrazu. Je to najmä kvôli zníženiu objemu obrazových dát, ktoré privádzame na vstup siete. V tejto kapitole si popíšeme základné metódy predspracovania obrazu. Vzorce a značenie použté v tejto kapitole je z [15].

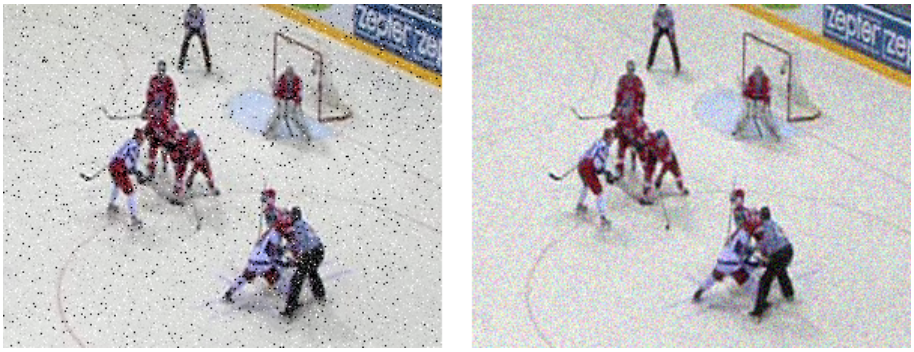
4.1 Šum

Šum v obraze si môžeme predstaviť ako náhodné chyby, rôzne odchýlky od pôvodnej obrazovej informácie. Do obrazu sa dostáva už pri jeho zachytávaní, alebo pri prenose. Šum je charakterizovaný určitou pravdepodobnostnou charakteristikou. Šum, v ktorom sú všetky frekvencie zastúpené rovnakou intenzitou, je nazvaný *biely šum.*

Špeciálnym typom bieleho šumu je Gaussovský šum. Je to veľmi dobrá aproximácia reálneho šumu. Hodnota náhodnej premennej je daná pravdepodobnosťou danou Gaussovskou krivkou. V jednorozmernom prípade je táto pravdepodobnosť definovaná ako:

$$p(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{\frac{-(x-\mu)^2}{2\sigma^2}} \tag{4.1}$$

kde μ je požadovaná stredná hodnota a σ je požadovaná smerodajná odchýlka.



Obr. 4.1: Vľavo obrázok s pridaným šumom typu peper a soľ šum, vpravo obrázok s pridaným Gaussovským šumom.

Pokiaľ šum nezávisí od obrazového signálu, nazveme ho *aditívny šum* a vyjadríme:

$$f(x,y) = g(x,y) + \nu(x,y) \tag{4.2}$$

kde f je výsledný zašumený obraz, g je pôvodný obraz a ν je pridaný šum.

Ak veľkosť šumu v danom bode obrazu závisí na hodnote jasu daného bodu, nazveme ho *multiplikatívny šum* a vyjadríme ako:

$$f(x,y) = g(x,y)\nu(x,y) \tag{4.3}$$

kde f je výsledný zašumený obraz, g je pôvodný obraz a ν je pridaný šum.

Špeciálny prípad šumu je *peper a soľ*, kde sú do obrazu pridávané biele a čierne body.

Pre vyjadrenie kvality zašumeného obrazu slúži veličina *pomer odstupu signálu od šumu. (signal to noise ration / SNR).* Je to podiel šumu k obrazu a je definovaný ako:

$$SNR = \frac{\sum_{(x,y)} \nu^2(x,y)}{\sum_{(x,y)} f^2(x,y)} \tag{4.4}$$

Príklad Gaussovského a šumu typu peper a soľ vidíme na obrázku 4.1. Šum môže spôsobiť problémy pri spracovávaní zašumeného obrazu. Ako odstrániť šum si popíšeme v podkapitole 4.3.2. Pokiaľ by systém, ktorý obrázky pre danú úlohu spracováva, bol voči šumu odolný, šum odstraňovať pochopiteľne nemusíme.

Obr. 4.2: Príklad problému pri zobrazovaní obrazu, ktorý vznikol prekladom dvoch polosnímok.

4.2 Prekladanie obrazu

Prekladanie obrazu sa používa najmä pri prenose televízneho signálu. V dobách CRT televízií, kde televízia obraz vykresľovala elektronickým lúčom do obrazovej matice, to riešilo problém postupného zhasínania obrazovky. Pri rýchlosti prehrávania 25snímkov za sekundu by pri vykresľovaní posledného riadku bol už horný riadok zhasnutý. Preto bolo nutné prekresľovať obraz rýchlejšie. Zdvojnásobenie vykresľovania by síce problém riešilo, ale zvyšný prenos dát sa nedal v reálnom čase spracovávať. Preto sa v súčasnej dobe stále používa prekladanie obrazu, kde sa vykresľujú dve polosnímky, zosnímané v polovičnom vertikálnom rozlíšení. Prvý sa vykreslí na každý párny riadok, druhý na každý nepárny riadok. Polosnímky sú vykresľované dvojnásobnou frekvenciou. Každá polosnímka je však zosnímaná v inom čase a pri prekladaní obrazu z rýchlej scény, prípadne z prelomu scén, vzniká problém, kde je vidno naraz dve rôzne snímky. Výsledok môžeme vidieť na obrázku 4.2.

Spracovávaním takýchto dát, napríklad vytvorením trénovacej množiny, kde na jednej snímke je objekt deformovaný prekladaním, na inom je v poriadku, by mohlo spôsobiť pri učení problémy. Preto je dobré sa problémov vznikajúcich pri prekladaní obrazu v rámci ďalšieho spracovania zbaviť. Podľa [1] sú metódy na potlačenie prekladania tieto:

Miešanie (blend). Obe polsnímky sú zväčšené na veľkosť obrazu a preložené cez

seba. Výhodou je jednoduchosť, nevýhodou je rozmazanie pri pohyblivých scénach. Špeciálnym typom je miešanie iba niektorých častí obrazu, to však neprináša oveľa lepšie výsledky.

Zahadzovanie. Zahadzujeme každú druhú snímku, zobrazujeme iba jedeu. Úplne tak odstránime prekladanie, avšak prídeme o obrazové informácie v druhej polsnímke.

Progresívne. Zvýšime počet snímok za sekundu a zobrazíme po sebe oba snímky. Výhodou je úplne odstránené prekladanie.

Progresívny sken. Analyzujeme obraz a prekladanie potláčame iba tam kde je viditeľné. Taktiež zvýšime počet snímok za sekundu. Táto metóda dáva veľmi dobré výsledky.

Pre potreby našej úlohy, detekcia hokejistov, bude najlepšie použiť zahadzovanie. Hokejisti sú stále v pohybe a skoro vždy by sa na nich prekladanie prejavilo. Výhodou zahadzovania je i to, že obraz bude mať polovičné vertikálne rozlíšenie a bude ho tak možné rýchlejšie spracovať.

4.3 Metódy predspracovania obrazu

Metódy predspracovania obrazu môžeme rozdeliť na:

- Bodové transformácie - berú do úvahy iba vlastnosti bodu samotného
- Lokálne transformácie - výsledok závisí od bodu a jeho okolia
- Globálne transformácie - výsledok závisí od celého obrazu

Všetky tieto transformácie majú na vstupe i na výstupe obrázok. Lokálne transformácie využívajú k výpočtu konvolúciu, ktorá sa používa aj pri výpočtoch v konvolučných neurónových sietiach popísaných v kapitole 3.

4.3.1 Bodové transformácie

Bodové transformácie fungujú tak, že vstupný obraz prechádzame bod po bode a na základe aktuálnej hodnoty jasu bodu/pixlu vypočítame hodnotu jasu vo výstupnom obraze.

Podľa [15] ich delíme na *jasové korekcie závislé na pozícii*, *geometrické transformácie* a *šedotónové transformácie*. Jasové korekcie berú do úvahy nielen jas vstupného bodu, ale i jeho pozíciu v obraze. Šedotónové transformácie transformujú jas

vstupného bodu bez ohľadu na jeho pozíciu. Takouto operáciou je napríklad *prahovanie*, kde výstupný pixel bude čierny, pokiaľ je jas vstupného pixlu pod prahovou hodnotou a biely v opačnom prípade. Matematicky je to transformácia τ jasu p vstupného bodu s mierkou $[p_0, p_k]$ do jasu q výstupného bodu s novou mierkou $[p_0, p_k]$. Teda $q = \tau(p)$. Otočením vstupnej a výstupnej mierky môžeme získať *negatív*, vhodnou transformačnou krivkou môžeme docieliť vylepšenie kontrastu, a tak zvýšime celkovú čitateľnosť obrázku.

Geometrické transformácie nám dovolia zmeniť vlastnosti obrazu, prípadne jeho deformácie. Ide najmä o zmenu veľkosti, rotáciu, skosenie alebo nápravu rôznych geometrických skreslení.

Geometrická transformácia je vektorová funkcia $\mathbf{T}$, ktorá vstupný pixel na súradniciach (x, y) mapuje na nové súradnice (x', y'). T je definovaná dvoma rovnicami:

$$x' = T_x(x, y), y' = T_y(x, y) \tag{4.5}$$

Geometrická transformácia sa skladá z dvoch krokov: z transformácie jednotlivých bodov a z interpolácie jasovej zložky z niekoľkých bodov. Interpolácia je nutná, pretože pri transformácii bodov je výsledok reálne číslo a výsledný bod nemusí ležať na obrazovej mriežke. Preto je potrebné získať hodnotu jasu pre odpovedajúci pixel výstupného obrazu.

Transformácia sa väčšinou aproximuje polynómom n-tého rádu, teda:

$$x' = \sum_{r=0}^{m} \sum_{k=0}^{m-r} a_{rk} x^r y^k \tag{4.6}$$

$$y' = \sum_{r=0}^{m} \sum_{k=0}^{m-r} b_{rk} x^r y^k \tag{4.7}$$

Často sa používajú tzv. bilineárne transformácie - rovnice 4.8 a 4.9 alebo afinné transformácie - rovnice 4.10 a 4.11. Pomocou afinných transformácii sa dajú dosiahnuť bežné geometrické transformácie akými sú napríklad zmena veľkosti (4.14, 4.15), otočenie o uhol θ (4.12, 4.13), skosenie alebo posun. Príklady vidíme na obrázku 4.3.

$$x' = a_0 + a_1 x + a_2 y + a_3 xy \tag{4.8}$$

$$y' = b_0 + b_1 x + b_2 y + b_3 xy \tag{4.9}$$

$$x' = a_0 + a_1 x + a_2 y \tag{4.10}$$

$$y' = b_0 + b_1 x + b_2 y \tag{4.11}$$

(a) Pôvodný obraz (b) Rotácia (c) Skosenie

Obr. 4.3: Príklady geometrických transformácii.

$$x' = x\cos\theta + y\sin\theta \tag{4.12}$$

$$y' = -x\sin\theta + y\cos\theta \tag{4.13}$$

$$x' = ax \tag{4.14}$$

$$y' = bx \tag{4.15}$$

Interpolácia jasovej zložky bodu, ktorý leží mimo obrazovú mriežku, je to vlastne konvolúcia, teda výpočet hodnoty z okolitých bodov. Bežne sa používa iba malé okolie bodu. Najjednoduchšou je aproximácia pomocou *najbližšieho suseda*, kde výstupný pixel bude mať jas najbližšieho bodu. Lepšou je *lineárna aproximácia*. kde sa využijú 4 body z okolia, ktorých hodnoty jasov sa lineárne skombinujú s tým, že zastúpenie je úmerné vzdialenosti k spracovávanému bodu. *Bikubická aproximácia* interpoluje funkciou v tvare mexického klobúka pomocou 16tich bodov z okolia. Pre jednorozmerný prípad je definovaná ako:

$$h = \begin{cases} 1 - 2|x|^2 + |x|^3 & \text{pre } 0 \leq |x| < 1 \\ 4 - 8|x| + 5|x|^2 - |x|^3 & \text{pre } 1 \leq |x| < 2 \\ 0 & \text{inak} \end{cases} \tag{4.16}$$

4.3.2 Lokálne transformácie

Pri lokálnej transformácii je hodnota jasu bodu vo výstupnom obraze vypočítaná z okolia bodu vo vstupnom obraze. Obyčajne ide o malé a štvorcové okolie bodu. Môžeme to rozdeliť na

- *vyhladzovanie* - odstránenie šumu, resp. potlačenie vysokých frekvencií.

- *gradientné operátory* - založené na derivácii gradientu obrazovej funkcie, tu patrí napríklad ostrenie, detekcia hrán a pod.

Podľa iného delenia delíme tieto transformácie na *lineárne* a *nelineárne*. Lineárny operátor definuje výstup ako lineárnu kombináciu bodov z okolia O. Ide o diskrétnu konvolúciu:

$$\mathbf{f} * \mathbf{k} = f(x,y) * k(x,y) = \sum_{(m,n)\in O} k(x-m, y-n) f(m,n) \tag{4.17}$$

kde $\mathbf{k}$ je jadro konvolúcie a voláme ho *konvolučná maska*.

Vyhladzovanie

Najjednoduchšie odstránenie aditívneho Gaussovského šumu je priemerovanie obrázkov. Pokiaľ máme iba jeden obrázok, tak priemer počítame z lokálneho okolia bodu. Výsledky priemerovania sú dobré na potlačenie šumu, bohužiaľ rozmazávajú hrany v obraze. Priemerovanie je možné zapísať pomocou konvolúcie. Kovnolučné jadro vyjadrené maticou vidíme na 4.18. Prípadne môžeme priemerovať so zvýrazneným stredom, čo lepšie zodpovedá vlastnostiam Gaussovského šumu. Konvolučné jadrá týchto transformácii sú 4.19 a 4.20. Výsledky zobrazuje obrázok 4.4.

$$\mathbf{k} = \frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \tag{4.18}$$

$$\mathbf{k} = \frac{1}{10}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 2 & 1 \\ 1 & 1 & 1 \end{bmatrix} \tag{4.19}$$

$$\mathbf{k} = \frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix} \tag{4.20}$$

V prípade, že chceme odstrániť šum, ale zachovať hrany, musíme použiť sofistikovanejšie riešenie. To nám poskytujú nelineárne transformácie. Jednou z nich je *metóda rotujúcej masky*, kde okolo bodu rotujeme rôznou maskou a vyberieme tú, ktorá má najmenší rozptyl.

Ďalšou nelineárnou transformáciou je filtrácia mediánom. Z okolia bodu vyberieme medián, ktorý bude výstupom. Medián znesie až 50 percent vychýlených hodnôt. Je to robustná metóda, ale nevýhodou je, že porušuje ostré rohy a tenké čiary v obraze.

(a) Pôvodný obraz (b) Zašumený obraz (c) Vyhladenie pomocou matice 4.18. (d) Vyhladenie pomocou matice 4.19. (e) Vyhladenie pomocou matice 4.20.

Obr. 4.4: Odstránenie šumu pomocou priemerovania.

Gradientné operátory

Medzi tieto operátori patria hranové detektory, teda operátory, ktoré zvýrazňujú hrany. Hrana je miesto v obraze, kde sa náhle mení hodnota jasu. Hrany sú do istej miery invariantné voči zmene osvetlenia. Preto sa často detektory alebo zvýrazňovače hrán používajú v počítačovom videní.

Hrana je daná vlastnosťami bodu a jeho okolia. Je to vektor s dvoma zložkami, veľkosťou a smerom. Existujú základné druhy hrán: skoková, strechová, tenká čiara avšak v reálnych obrazoch tieto ideálne druhy hrán nenachádzame, ale máme iba rôzne zašumené hrany.

Existujú tri druhy hranových detektorov:

- hľadanie maxím prvej derivácie
- hľadanie prechodov nulou u druhých derivácii
- lokálnou aproximáciou obrazovej funkcie parametrickým modelom.

V prípade spojitého dvojrozmerného obrazu je veľkosť gradientu $||\nabla f(x,y)||$ a smer ψ definovaný ako:

$$||\nabla f(x,y)|| = \sqrt{\left(\frac{\partial f}{\partial x}\right)^2 + \left(\frac{\partial f}{\partial y}\right)^2}, \text{pokiaľ sú delitele nenulové} \tag{4.21}$$

$$\psi = \arctan\left(\frac{\partial f}{\partial x} / \frac{\partial f}{\partial y}\right), \text{pokiaľ sú delitele nenulové} \tag{4.22}$$

Derivácia $f(x,y)$ vo smere (u,v) je:

$$(u, v) \cdot \nabla f(x, y) = \left(u\frac{\partial f}{\partial x}, v\frac{\partial f}{\partial y} \right) \tag{4.23}$$

Pretože obraz nie je spojitý, aproximujú sa derivácie konečnými diferenciami. Pre jednorozmerný prípad platí:

Nesymetrická, (v bode i - 1/2) je:

$f'(i) \approx f(i) - f(i-1)$,

Symetrický tvar:

$f'(i) = f(i+1) - f(i-1)$

Ako príklad uvedieme konvolučné masky hranových detektorov:

Robertsov operátor:

$$\mathbf{k}_1 = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \qquad \mathbf{k}_2 = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix} \tag{4.24}$$

Prewitt operátor:

$$\mathbf{k}_1 = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix}, \qquad \mathbf{k}_2 = \begin{bmatrix} 0 & 1 & 1 \\ -1 & 0 & 1 \\ -1 & -1 & 0 \end{bmatrix}, \qquad \ldots \tag{4.25}$$

Sobelov operátor:

$$\mathbf{k}_1 = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}, \qquad \mathbf{k}_2 = \begin{bmatrix} 0 & 1 & 2 \\ -1 & 0 & 1 \\ -2 & -1 & 0 \end{bmatrix}, \qquad \ldots \tag{4.26}$$

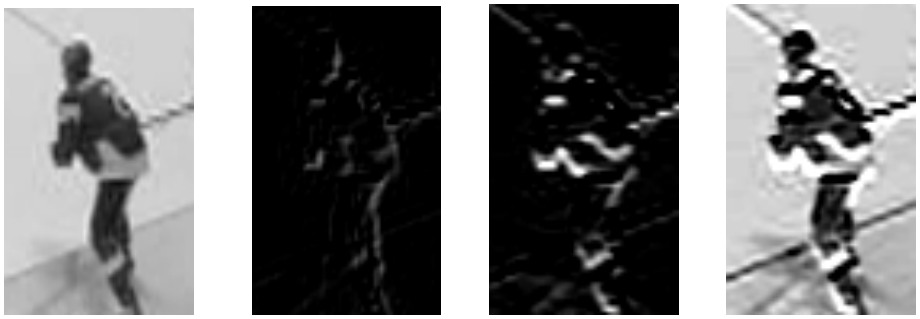
Kirschov operátor:

$$\mathbf{k}_1 = \begin{bmatrix} 3 & 3 & 3 \\ 3 & 0 & 3 \\ -5 & -5 & -5 \end{bmatrix}, \qquad \mathbf{k}_2 = \begin{bmatrix} 3 & 3 & 3 \\ -5 & 0 & 3 \\ -5 & -5 & 3 \end{bmatrix}, \qquad \ldots \tag{4.27}$$

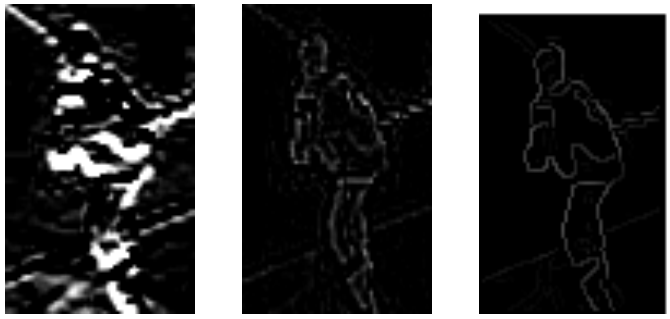
Pri operátorchoch, ktoré hľadajú priechody druhej derivácie gradientu nulou, vypočítame druhú deriváciu ako zloženie prvých derivácii, teda:

$$\frac{\partial^2}{\partial x^2} \approx [-1, +1] * [-1, +1] = [+1, -2, +1] \tag{4.28}$$

Diskrétny laplacián je súčtom druhých parciálnych derivácii



(a) Pôvodný obraz (b) Robertsov operátor (c) Perwitt operátor (d) Sobelov operátor



(e) Kirschov operátor (f) Druhá derivácia (g) Cannyho detektor

Obr. 4.5: Hranové detektory.

$$\nabla^2 \approx \begin{bmatrix} 0 & 0 & 0 \\ 1 & -2 & 1 \\ 0 & 0 & 0 \end{bmatrix} + \begin{bmatrix} 0 & 1 & 0 \\ 0 & -2 & 0 \\ 0 & 1 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix} \tag{4.29}$$

Hranovým detektorom, ktorý je v dnešnej dobe používaný vo väčšine aplikácii, je Cannyho detektor. Jeho algoritmus je nasledujúci:

1. Nájdi približné smery gradientu.
2. Pre každý pixel nájdi 1D deriváciu v smere gradientu pomocou 'optimálnej' masky spojujúcej vyhladenie a deriváciu.
3. Nájdi lokálne maximá týchto derivácií.
4. Hranové body získaj prahovaním s hysterézou.
5. Urob syntézu hrán získaných pre rôzne veľké vyhladenia (málokedy sa používa)

Detekcie hrán jednotlivých hranových detektorov zobrazuje obázok 4.5.

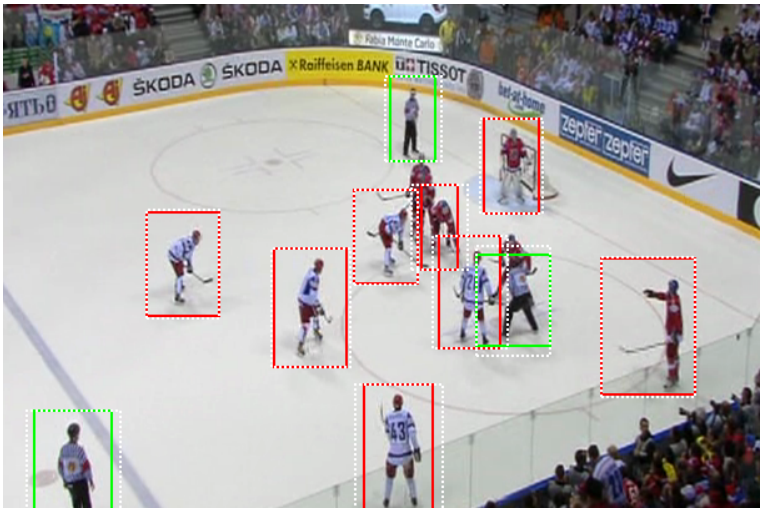
V tejto kapitole sme si popísali základy predspracovania obrazu, priblížili metódy odstránenia šumu a detekcie hrán. V prípade spracovania obrazu pomocou klasických neurónových sietí by sme pre zníženie veľkosti vstupného vektoru museli napríklad v obraze nájsť hrany a pomocou nich obraz popísať. To ale v prípade použitia konvolučných sietí nemusíme urobiť. Tieto poznatky nám však dovolia porozumieť aké príznaky sa daná konvolučná sieť naučila detekovať. Využijeme to neskôr v kapitole 6.

Kapitola 5

Výber modelu

Ako cieľ práce sme si vybrali detekciu hokejových hráčov na snímkach, prípadne z videa hokejového zápasu. Výstupom by mal byť teda detektor, ktorý na vstupných obrázkoch označí jednotlivých hokejistov. Využitie vidíme napríklad v použití tzv. *rozšírenej reality*, kde môžeme do pôvodného obrazu k jednotlivým hokejistom dokresľovať rôzne informácie, ako je napríklad štatistika tímu, hokejistu a pod. Pre riešenie tejto úlohy sme si vybrali konvolučné siete, a to najmä kvôli tomu, že pracujú priamo s obrazom a príznaky, z ktorých sieť pozná, že sa jedná o hokejistu, si sieť „nájde“ sama. Keby sme chceli použiť klasické neurónové siete, museli by sme vstupný obrázok predspracovať a získať z neho lokálne príznaky, ako sú napríklad hrany, rozmery, farbu a pod, navyše tieto príznaky by mali byť invariantné voči rôznym geometrickým skresleniam. Vybrať takéto príznaky zo vstupných vzorov býva veľmi náročné. V prípade konvolučných sietí toto robiť nemusíme a správne navrhnutá konvolučná sieť sa o všetko postará sama. Ako sme popísali v kapitole 3, sú konvolučné siete invariantné voči posunu objektu. To sa nám pri návrhu detektora môže hodiť, pretože potom nemusíme prechádzať každý výrez vstupného obrázku, ale stačí nám ho posúvať vo väčších krokoch. V tejto kapitole navrhneme rôzne konvolučné siete, ktoré naučíme a otestujeme. Potom podľa vopred stanovených priorít jednu z nich vyberieme a použijeme pri implementácii detektora.

Detektor by mal byť dostatočne robustný a nemalo by ho spliesť rôzne natočenie a poloha hráčov a v ideálnom prípade by mu nemalo robiť problém detektovať hráčov v rôznych dresoch. Hokejové zápasy sa väčšinou konajú pod zastrešeným a umelo osvetleným štadiónom, do veľkej miery preto odpadávajú problémy s rôznym osvetlením scény. Na obrázku 5.1 vidíme scénu, na ktorej sa budeme snažiť hráčov detektovať.



Obr. 5.1: Typická scéna z hokejového zápasu s vyznačením hokejistov.

5.1 Príprava dát

Konvolučné neurónové siete budeme učiť metódou s učiteľom, preto je potrebné získať trénovacie dáta. V našom prípade ide o vytvorenie siete, ktorej na vstup privedieme obrázok a sieť nám odpovie, či ide o hráča, alebo o pozadie. Musíme pripraviť dostatok výrezov hráčov a pozadí aby sme sieť mohli dobre naučiť. Vzorky hráčov by mali pokryť čo najviac polôh hráčov a čo najviac farieb a variánt dresov. V ďalšom texte si povieme, ako sme pripravili dáta a čo sme z nich vyčítali a použili pri návrhu konvolučných sietí.

Ako zdroj dát boli použité online záznamy z hokejových majstrovstiev sveta 2011 a 2012 dostupných počas šampionátu. Z týchto záznamov sme vybrali úryvky z niekoľkých zápasov, v ktorých hrali družstvá s rôznymi farbami dresov. Úryvky sme previedli na jednotlivé obrázky a z nich vybrali zábery na hraciu plochu zboku. Ručne s pomocou vlastného programu *Označ Hráčov* popísaného v užívateľskej dokumentácii sme označili jednotlivých hokejistov a rozhodcov. Program nám z obrázkov vyrezal označených hráčov, tých sme potom použili na učenie siete. Výrezy pozadia neobsahujúce hokejistov nám program vygeneroval sám. Na snímkach sme vyznačili dohromady 4200 hokejistov. Príklady týchto výrezov vidíme na obrázku 5.2.

Vzorky pochádzajú z niekoľkých zápasov a to konkrétne:

- Česko - Rusko: farby dresov červená a biela - 2454 vzorov,
- Švédsko - Česko: farby dresov žlto-modrá a červená - 242 vzorov,
- Švédsko - Fínsko: farby dresov žlto-modrá a modrá - 1232 vzorov,
- Rôzne zápasy a farby dresov: 261 vzorov - táto množina bude slúžiť iba na testovanie.

Obr. 5.2: Ukážka vzorov z jednotlivých množín. Prvý riadok ukazuje hráčov, druhý pozadia.

	x	y	šírka	výška	pomer výška/šírka
priemer	330	224	71	116	1,68
medián	335	214	68	109	1,63
minimum	0	0	24	56	0,79
maximum	667	495	364	503	3,45

Tabuľka 5.1: Tabuľka ukazujúca rozloženie veľkosti a polohy skúmaných vzorov.

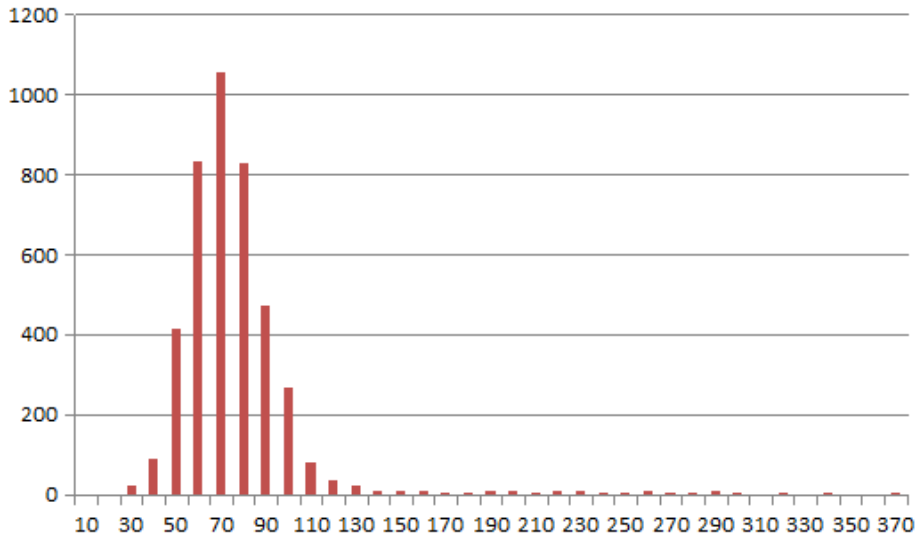
Tieto výrezy analyzujeme, aby sme zistili, aký veľký vstup je najlepšie sieti predložiť.

Analyzovali sme najbežnejšiu šírku hráča, pomer strán výrezu a jeho pozíciu v obraze. Analýzou šírky a pomeru strán dosiahneme najlepšiu veľkosť vstupu pre neurónovú sieť. Analýza pozície nám môže pomôcť pri návrhu detektora aby sme nemuseli prechádzať úplne celý obraz.

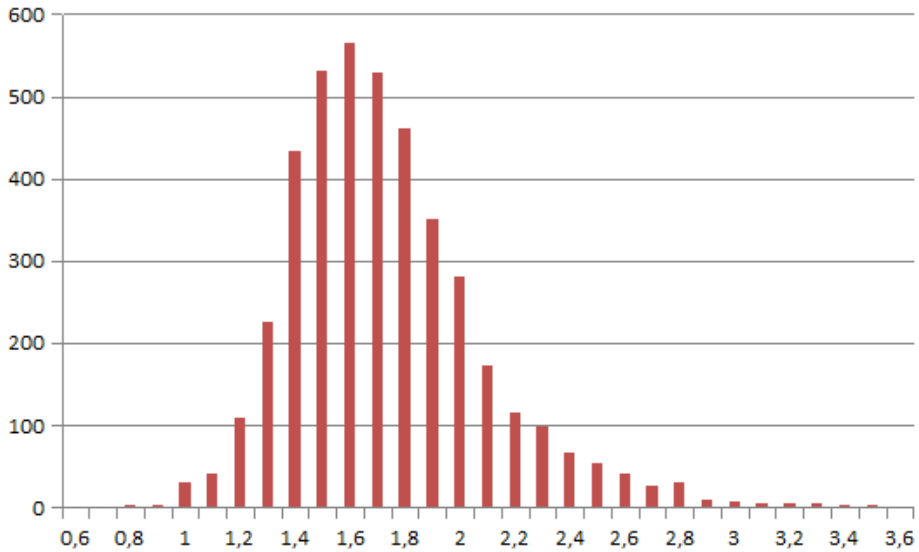
Histogram rozloženia šírok vyrezaných hráčov vidíme na obrázku 5.3. Najčastejšia šírka je 70 px (spomenieme, že analyzujeme výrezy z obrázkov o veľkosti 720 px×576 px, teda v rozlíšení podľa televíznej normy PAL), čo je cca 10 % z celkovej šírky. Pri pohľade na obrázok 5.4, ktorý zobrazuje histogram pomerov strán, vidíme, že najbežnejší pomer strán šírka k výške je 1,6. Najlepší vstup pre sieť by teda bol obrázok s šírkou 70 px a výškou 112 px.

Na obrázku 5.5 vidíme rozloženie jednotlivých hokejistov v priestore. Je zrejmé, že sa vyskytujú po celej hracej ploche a nie sú tam žiadne prázdne miesta (prázdne miesto na spodku scény je spôsobené tým, že body zobrazujú pravý horný roh hráča. Hráč teda zasahuje do priestoru napravo a pod bodom). Najviac hokejistov sa vyskytuje v strede scény. Detektor bude musieť preskúmať celý obrázok.

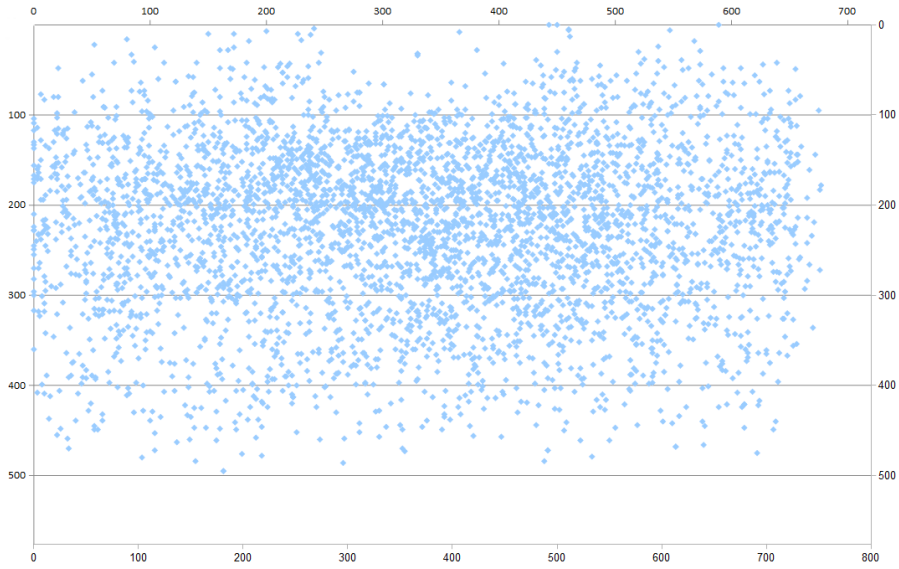
Tabuľka 5.1 ukazuje priemerné hodnoty, medián a medze skúmaných hodnôt.



Obr. 5.3: Rozloženie šírok jednotlivých hráčov. Najpočetnejšiu skupinu tvoria hráči so šírkou 70–80 px.



Obr. 5.4: Rozloženie pomeru výšky k šírke jednotlivých hráčov. Najpočetnejšiu skupinu tvoria hráči s pomerom strán 1,6.



Obr. 5.5: Rozloženie pozície (zobrazený je ľavý horný roh hráča) jednotlivých hráčov v celom obraze.

5.2 Požiadavky

Pred samotným návrhom konvolučnej siete pre našu úlohu si zhrňme, aké vlastnosti od nej očakávame a podľa čoho budeme viaceré navrhnuté siete porovnávať.

Naša práca spočíva v tom, že na zadanom vstupnom obrázku, prípadne videu, nájdeme a označíme hľadané objekty. V našom prípade je na vstupnom obrázku scéna, ktorá zachytáva hraciu plochu hokejového štadióna, na ktorom práve prebieha zápas. Detekovanými objektami sú v tomto prípade hráči. Detekcia hráčov by mala prebiehať tak, že sa sieti postupne predkladajú výrezy scény a sieť by odpovedala, či je na danom mieste hráč, alebo pozadie. Všetko by malo prebiehať bez predspracovania. Sieť slúži ako klasifikátor priamo z obrazových dát.

Vo vyššie popísanom prípade pôjde najmä o rýchlosť rozpoznávania, pretože by to v ideálnom prípade malo prebiehať v reálnom čase. Keďže sa snímky budú získavať pravdepodobne z televízneho vysielania, je potrebné aby bola sieť odolná voči šumu, ktorý sa môže vyskytovať pri príjme slabšieho signálu. V prípade, že by sme chceli mať detektor implementovaný priamo v televízore, kde môže byť výpočtová kapacita obmedzená, je rýchlosť rozpoznania dôležitým faktorom. Z podobného dôvodu môže byť obmedzená kapacita pamäte a preto je dobré, aby sieť potrebovala pre svoj výpočet pamäte menej.

Pamäťová náročnosť klasických neurónových sieti je dosť vysoká, každý neurón má nezávislé parametre a s narastajúcim počtom neurónov stúpa počet týchto parametrov spolu s pamäťovými nárokmi. V prípade konvolučných sieti sa využíva tzv. zdieľanie parametrov, teda niekoľko stoviek neurónov na jednej ploche zdieľa

rovnaké parametre a tým rapídne znižuje pamäť potrebnú k uchovaniu siete. Konvolučné siete však môžu obsahovať oveľa viac výpočtových jednotiek a tým zvyšovať výpočtovú náročnosť. To sa dá zmierniť napríklad paralelným rozdelením výpočtu. Ako príklad uvedieme, že klasická neurónová sieť, ktorej vstupom by bol obrázok veľkosti 30×45 px s jednou skrytou vrstvou o veľkosti 1000 neurónov a 2 výstupnými neurónmi, má cca 3000000 parametrov, konvolučná sieť s rovnakým vstupom by mala niekoľko desiatokkrát menej parametrov.

Detektor bohužiaľ nemôžeme učiť všetkými možnými druhmi hráčov a ich dresov, preto ako ďalší porovnávací parameter pridáme schopnosť siete správne rozpoznávať hráčov v dresoch (farbách), na ktoré sieť nebola učená. Je potrebné predložiť sieti počas učenia čo najviac vzorov - s tým súvisí rýchlosť učenia. V našom prípade ide o 4000-5000 vzorov hráčov a rovnaký počet pozadí. Máme dohromady približne 9000 vzorov, z ktorých sa sieť musí naučiť. Tento proces môže trvať v závislosti na veľkosti vzoru, veľkosti siete a výpočtového výkonu dlho, až niekoľko dní. Pri navrhovaní parametrov a architektúry siete je nutné proces učenia opakovať a je výhodnejšie, ak netrvá dlho. Po zvolení konečnej architektúry a naučení siete nie je tento porovnávací parameter až taký dôležitý.

Zhrnieme si teda jednotlivé požiadavky na siete, podľa ktorých ich budeme porovnávať:

- úspešnosť klasifikácie na testovacej množine - *hlavný faktor*
- rýchlosť výpočtu - klasifikácie - *dôležité pri rozpoznávaní v reálnom čase*
- odolnosť voči šumu - *signál a obraz nemusí byť vždy čistý*
- rozpoznávanie hráčov v neznámych farbách dresov
- rýchlosť učenia.

5.3 Návrh architektúry siete

Analyzovaním označených hráčov sme zistili, že najvhodnejšia veľkosť vstupu (z pohľadu najmenšej straty obrazovej informácie) pre neurónovú sieť pri detekovaní hokejistov z obrázku o veľkosti 720×576 px je približne 70×110 px (šírka ×výška). Sieť by mala vedieť rozpoznávať i hráčov s farbami dresov, ktoré „ešte nevidela“. Pri učení farebných obrázkov je možné, že by sa sieť naučila iba natrénované farby, vytvoríme preto taktiež siete, ktoré budú mať na vstupe obrázok čiernobiely.

Navrhli sme konvolučnú neurónovú sieť *Net1.6-70* so vstupným obrázkom s veľkosťou 70×110 px s parametrami, ktoré vidíme v tabuľke 5.2.

Predtým, ako začneme túto sieť učiť, zistíme si aká je jej rýchlosť klasifikácie. Spočítajme si koľko výrezov budeme musieť predložiť sieti, aby spracovala jednu snímku.

typ vrstvy	počet máp	konvolučné jadro	veľkosť mapy
vstup	1 alebo 3[1]		70×110 px
konvolučná	8	7×7	64×104
subsamplingová	8	4×4	16×26
konvolučná	64	5×5	12×22
subsamplingová	64	3×2	4×11
konvolučná	128	3×3	2×9
subsamplingová	128	2×3	1×3
neurónová	50 neurónov		
výstup	2 neuróny		

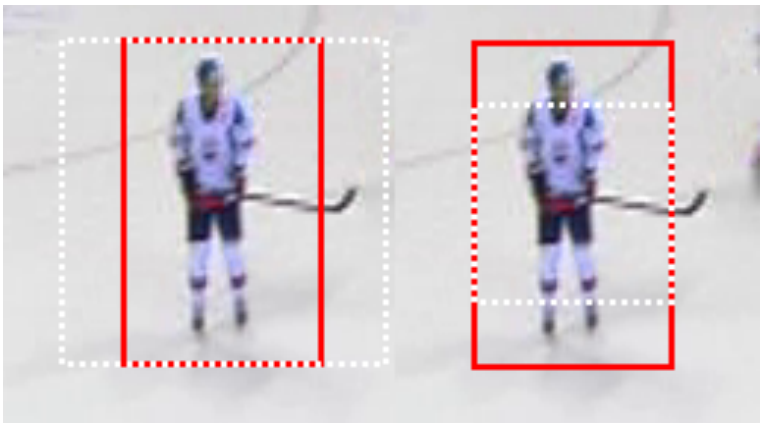
Tabuľka 5.2: Parametre konvolučnej siete *Net1.6-70* so vstupom o veľkosti 70×110 px

Modelovú scénu o veľkosti 720×576 px budeme prechádzať obdĺžnikom s veľkosťou rovnou veľkosti vstupu navrhnutej siete, teda 70×110 px. Keďže konvolučné neurónové siete sú odolné voči posunu, nemusíme kontrolovať každý výrez. Budeme sa posúvať v krokoch veľkých 20 % daného rozmeru vstupu, teda v x-ovej osi po 14 px a v y-ovej osi po 22 px. Týmto prechodom musíme otestovať 47×22=1034 vzorov. Nie všetci hráči sú však rovnako veľkí, prejdeme teda obrázok i s výrezom väčším a výrezom menším, tento výrez potom upravíme na veľkosť vstupu siete. Pre detekciu jedného obrázku potrebujeme skontrolovať približne 2000-4000 vzorov, v závislosti na tom, koľko veľkostí chceme vyskúšať.

Navrhnutá neurónová sieť zvláda otestovať 4000 vzorov na testovanej konfigurácii[2] za 19 sekúnd. To je ale pre našu úlohu priveľa. Navrhli sme preto sieť *Net1.6-35* s menším vstupom. Jej parametre vidíme v tabuľke 5.3. Táto sieť má na vstupe dvakrát menší obrázok ako prvá navrhnutá sieť. To zaručuje rýchlejšie vyhodnocovanie siete. Otestovanie 4000 vzorov zvládla táto menšia sieť za 4,5 sekundy, čo je už prijateľnejšie. Pre tretiu navrhnutú sieť *Net1.6-15* sme zvolili ešte menší rozmer vstupu a to 15 ×24 px. Jej parametre vidíme v tabuľke 5.4. Táto sieť zvláda otestovať 4000 vzorov za 1,7 sekundy.

Navrhli sme siete *Net1.6-70*, *Net1.6-35* a *Net1.6-15*, ktoré majú na vstupe obrázok s pomerom strán výšky k šírke 1,6. Keďže konvolučné siete typicky pracujú so štvorcovým vstupom, navrhneme niekoľko sieti práve s ním. Štvorec môžeme výrezu buď opísať alebo vpísať. Opísaný štvorec zahrnie i okolie, ktoré môže pomôcť pri rozpoznávaní o aký objekt ide, vpísaný naopak vyreže stred výrezu a sieť sa môže zamerať iba na detaily. Vpísaný a opísaný štvorec môžeme vidieť na obrázku 5.6. Parametre sietí so štvorcovým vstupným obrázkom *NetSquare36* a *NetSquare16* vidíme v tabuľkách 5.5 a 5.6.

[2]Intel® Core™ i5-2450M (dva procesory, štyri vlákna, 2,5 – 3,1 GHz), pomocou knižnice *Torch 7*, systém Ubuntu 10.4 64bit



Obr. 5.6: Červenou čiarou sme ručne označili hokejistu, prerušovanou čiarou je vyznačený výrez, ktorý budeme predkladať sietiam so štvorcovým vstupom. Naľavo je opísaný štvorec, napravo je vpísaný.

typ vrstvy	počet máp	konvolučné jadro	veľkosť mapy
vstup	1 alebo 3		36×58 px
konvolučná	8	7×7	30×52
subsamplingová	8	3×4	10×13
konvolučná	64	5×5	6×9
subsamplingová	64	3×2	2×3
konvolučná	128	2×3	1×1
neurónová	50 neurónov		
výstup	2 neuróny		

Tabuľka 5.3: Parametre konvolučnej siete *Net1.6-36* so vstupom o veľkosti 36×58 px

typ vrstvy	počet máp	konvolučné jadro	veľkosť mapy
vstup	1 alebo 3		15×24 px
konvolučná	8	4×4	12×21
subsamplingová	8	3×3	4×7
konvolučná	128	3×3	2×5
subsamplingová	128	2×5	1×1
neurónová	50 neurónov		
výstup	2 neuróny		

Tabuľka 5.4: Parametre konvolučnej siete *Net1.6-15* so vstupom o veľkosti 15×24 px

typ vrstvy	počet máp	konvolučné jadro	veľkosť mapy
vstup	1 alebo 3		36×36 px
konvolučná	8	7×7	30×30
subsamplingová	8	3×3	10×10
konvolučná	64	5×5	6×6
subsamplingová	64	3×3	2×2
konvolučná	128	2×2	1×1
neurónová	50 neurónov		
výstup	2 neuróny		

Tabuľka 5.5: Parametre konvolučnej siete *NetSquare36* so vstupom o veľkosti 36×36 px

typ vrstvy	počet máp	konvolučné jadro	veľkosť mapy
vstup	1 alebo 3		16×16 px
konvolučná	8	3×3	14×14
subsamplingová	8	2×2	7×7
konvolučná	64	3×3	5×5
subsamplingová	64	5×5	1×1
neurónová	50 neurónov		
výstup	2 neuróny		

Tabuľka 5.6: Parametre konvolučnej siete *NetSquare16* so vstupom o veľkosti 16×16 px

5.4 Testovanie navrhnutých architektúr

V predchádzajúcej podkapitole sme navrhli niekoľko architektúr konvolučných sietí. Siete sa líšili veľkosťou vstupu, farebnosťou vstupného obrázku a počtom a parametrami ďalších vrstiev. Prvá z nich s ideálnym (vzhľadom na stratu obrazových informácii) rozmerom vstupu (*Net1.6-70*) bola veľká a vyhodnocovanie trvalo príliš dlho, použitie tejto siete rovno zamietneme. Ďalšie dve siete *Net1.6-35* a *Net1.6-15* sú už rýchlejšie. Ďalej sme vytvorili siete so štvorcovým vstupom *NetSquare36* a *NetSquare16*. Ako sme už napísali vyššie, budeme testovať presnosť učenia na testovacej množine, rýchlosť vyhodnocovania a úspešnosť vyhodnocovania hráčov v neznámych dresoch. Vytvorili sme tieto skupiny vzorov:

- *Trénovacia množina* - obsahuje cca 3500 výrezov hráčov a rovnako výrezov pozadí.
- *Testovacia množina 1* - obsahuje cca 350 výrezov hráčov a rovnako výrezov pozadí. Výrezy sú z rovnakých zápasov ako trénovacia množina.
- *Testovacia množina 2* - obsahuje cca 260 výrezov hráčov z odlišných zápasov. Táto množina slúži na overenie úspešnosti vyhodnocovania hráčov v neznámych dresoch.

5.4.1 Rýchlosť

Rýchlosť sietí pri rozpoznávaní je úmerná zložitosti siete. Testovali sme rýchlosť učenia i testovania na notebooku v konfigurácii Intel® Core™ i5-2450M (dva procesory, štyri vlákna, 2,5 – 3,1 GHz), pomocou knižnice *Torch 7* v systéme Ubuntu 10.4 64bit. Konkrétne hodnoty zobrazuje tabuľka 5.7. Siete s menším vstupom rozpoznávajú rýchlejšie. Pretože potrebujeme rýchle rozpoznávanie, uprednostníme menšie siete.

5.4.2 Presnosť na testovacích množinách

Každú sieť sme natrénovali a otestovali na vyššie uvedených množinách. Najprv pomocou farebných vzorov a potom pomocou vzorov čiernobielych. Siete so štvorcovým vstupom sme učili s vypísanými i opísanými štvorcovými výrezmi. Nechali sme ich učiť po 400 iteráciách, každú sieť trikrát. Priemerné výsledky vidíme v tabuľke 5.8. Tučne sú označené tri najlepšie hodnoty pre danú množinu. Ako môžeme vidieť, tak siete so štvorcovým vstupom mali nižšiu úspešnosť na trénovacích množinách ako siete so vstupom obdĺžnikovým. To je pravdepodobne spôsobené tým, že do opísaného štvorca zasahujú iné objekty ako hokejisti a sieť určí takéhoto hokejistu za

sieť	vstup	čas trénovania 1000 vzorov (1 iterácia)	čas otestovania 1000 vzorov
Net1.6-15	far.	898 ms	361 ms
Net1.6-15	čb.	873 ms	283 ms
Net1.6-36	far.	5542 ms	1677 ms
Net1.6-36	čb.	4141 ms	1220 ms
NetSquare16	far. opis.	781 ms	353 ms
NetSquare16	far. vpis.	740 ms	296 ms
NetSquare16	čb. opis.	699 ms	289 ms
NetSquare16	čb. vpis.	655 ms	311 ms
NetSquare36	far. opis.	3974 ms	1163 ms
NetSquare36	far. vpis.	3670 ms	1068 ms
NetSquare36	čb. opis.	2737 ms	858 ms
NetSquare36	čb. vpis.	2857 ms	913 ms

Tabuľka 5.7: Rýchlosť trénovania a testovania jednotlivých sietí.

pozadie. Naopak pri vpísaných štvorcoch je možné, že hokejista nebol dobre vycentrovaný a vpísaný štvorec obsahuje z väčšej časti ľadovú plochu. „Farebné siete“ mali oproti „čiernobielym“ úspešnosť vyššiu.

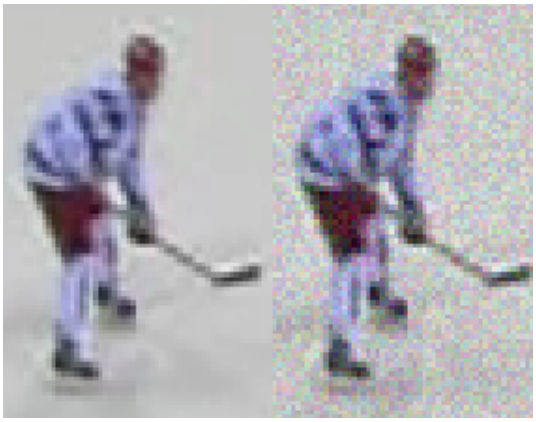
5.4.3 Odolnosť voči šumu

Testovali sme i odolnosť voči šumu. *Testovaciu množinu 1* sme naklonovali 100 krát a rôzne zašumeli Gaussovským šumom so strednou hodnotou 0 a rozptylom 0,1 (vstupné obrázky boli normované do intervalu (0, 1)). Obrázok 5.7 ukazuje príklad zašumeného vzoru. Naučené siete sme nechali tieto množiny vyhodnotiť. V tabuľke 5.8 vidíme priemernú úspešnosť (zo sto rozpoznávaní testovacej množiny) jednotlivých sietí. Navrhnuté konvolučné siete sú odolné voči šumu. V určitých prípadoch nastalo, že chyba pri zašumených vzoroch bola o trochu menšia, ako pri nezašumených, to môže byť spôsobené tým, že šum vyzdvihol nejaké príznaky, podľa ktorých sa sieť pri vyhodnocovaní rozhoduje.

5.5 Výber

Siete *Net1.6-15* a *Net1.6-36* sú na tom na testovacích množinách podobne a preto môžeme uprednostniť kvôli rýchlosti menšiu sieť *Net1.6-15*. Túto sieť s predkladaním farebných výrezov použijeme pri detekovaní hokejistov.

Pre implementovanie detektora sme si vybrali sieť *Net1.6-15* s farebným vstupom, ktorá mala veľmi dobrú úspešnosť na testovacích množinách a je dostatočne malá



Obr. 5.7: Vľavo nezašumený vzor, vpravo zašumený.

sieť	vstup	úspešnosť na trén. množ.	úspešnosť na test. množ. 1	úspešnosť na test. množ. 2	úspešnosť na zašumenej množine
Net1.6-15	far.	99,65 %	**98,19 %**	**100 %**	**98,03 %**
Net1.6-15	čb	99,19 %	97,51 %	**99,6 %**	97,48 %
Net1.6-36	far.	**99,88 %**	**97,96 %**	**99,8 %**	**98,00 %**
Net1.6-36	čb	**99,81 %**	**98,07 %**	**100 %**	**98,06** %
NetSquare16	far. opis.	99,10 %	96,29 %	99,39 %	96,24 %
NetSquare16	far. vpis.	99,32 %	94,32 %	**100 %**	94,54 %
NetSquare16	čb opis.	98,27 %	95,98 %	98,99 %	95,89 %
NetSquare16	čb vpis.	98,77 %	94,14 %	**99,6 %**	94,24 %
NetSquare36	far. opis.	**99,75 %**	97,08 %	**100 %**	97,12 %
NetSquare36	far. vpis.	**99,75 %**	95,18 %	**100 %**	95,07 %
NetSquare36	čb opis.	99,67 %	97,4 %	99,39 %	97,37 %
NetSquare36	čb vpis.	99,73 %	94,99 %	**99,8 %**	94,96 %

Tabuľka 5.8: Úspešnosť jednotlivých sietí (zobrazená je priemerná hodnota z troch behov učenia/testovania) na trénovacej a testovacích množinách. Tučne sú označené tri najlepšie hodnoty pre každú množinu (stĺpec). Vidíme, že štvorcové siete na testovacích množinách nemajú veľmi dobrú úspešnosť. Rovnako môžeme pozorovať, že siete s farebným vstupom sú na tom lepšie. Najlepšie z porovnania vychádza sieť *Net1.6-15* s farebným vstupom, pretože je malá a teda rýchla a má i dobré úspešnosti na testovacích množinách. Posledný stĺpec ukazuje priemernú úspešnosť (priemer zo 100 behov) na zašumenej testovacej množine 2.

na to, aby dokázala rýchlo vyhodnocovať predložené výrezy. Nerobia jej problém ani zašumené vzory.

5.5.1 Optimalizácia parametrov vybranej architektúry

V predchádzajúcom texte sme zistili, že na detekciu hráčov bude stačiť relatívne malá sieť so vstupom o veľkosti 15 px ×24 px. Pri návrhu konvolučných neurónových sietí je potrebné stanoviť veľa parametrov. Ide napríklad o počet príznakových máp v jednotlivých vrstvách, či o veľkosť konvolučných jadier v jednotlivých mapách. Stanovili sme si, že nám ide o úspešnosť rozpoznávania, ale i o rýchlosť. Rýchlosť je priamo spojená s počtom parametrov, ktoré musíme počas trénovania naučiť. Rozhodli sme sa preto vyskúšať rôzne počty príznakových máp v jednotlivých konvolučných vrstvách. Budeme trénovať a testovať sieť *Net1.6-15* s upravenými počtami príznakových máp a to:

- V prvej konvolučnej a subsamplingovej vrstve vyskúšame 2, 4, 6 a 8 príznakových máp.
- V druhej konvolučnej a subsamplingovej vrstve vyskúšame 32, 64, 96 a 128 príznakových máp.

Priemerné úspešnosti na testovacej množine 1 z troch rôznych behov trénovania vidíme v tabuľke 5.9. I keď je úspešnosť vo všetkých kombináciách parametrov podobná, je vidno, že 4 príznakové mapy v prvej konvolučnej vrstve sú pre našu úlohu ideálne, keďže 3 zo 4 pokusov s týmto počtom malo na testovacej množine najlepšiu úspešnosť. V druhej kovolučnej vrstve je optimálna hodnota 32 príznakových máp. Túto sieť budeme označovať ako *Net1.6-15-4-32* a použijeme ju ďalej pre detekovanie hokejistov.

Pre našu úlohu, detekovať hokejistov na ľadovej ploche, sa ukázala byť najlepšia sieť, ktorá ma na vstupe malý obrázok o veľkosti 15×24 px. Je dostatočne malá, aby dokázala rýchlo vyhodnocovať predložené vzory. Jej úspešnosť na testovacích množinách bola porovnateľná so sieťami, ktoré na vstupe predpokladali väčší obrázok. Tieto siete však sú pri vyhodnocovaní pomalšie. Nami vybraná sieť má dobré vlastnosti i pri rozpoznávaní zašumených vstupov.

počet máp v 1. konv. vrstve \ počet máp v 2. konv. vrstve	32	64	96	128
2	98,58 %	98,54 %	98,70 %	98,28 %
4	98,95 %	98,42 %	98,70 %	98,82 %
6	98,62 %	98,42 %	98,58 %	98,74 %
8	98,50 %	98,34 %	98,54 %	98,54 %

Tabuľka 5.9: Úspešnosti siete *Net1.6-15* s určitým počtom príznakových máp v prvej a druhej konvolučnej vrstve. Riadky určujú počet máp v prvej, stĺpce v druhej konvolučnej vrstve. Hodnoty v tabuľke sú priemerné hodnoty úspešnosti na testovacej množine 1 z troch behov učenia siete s danými parametrami. Najlepšie 4 hodnoty sú označené zelenou farbou, ďalšie 4 žltou, potom oranžovou a najhoršie 4 hodnoty sú označené červenou farbou. Môžeme vidieť, že ideálny počet príznakových máp v prvej konvolučnej vrstve je 4.

Kapitola 6

Analýza vybraného modelu

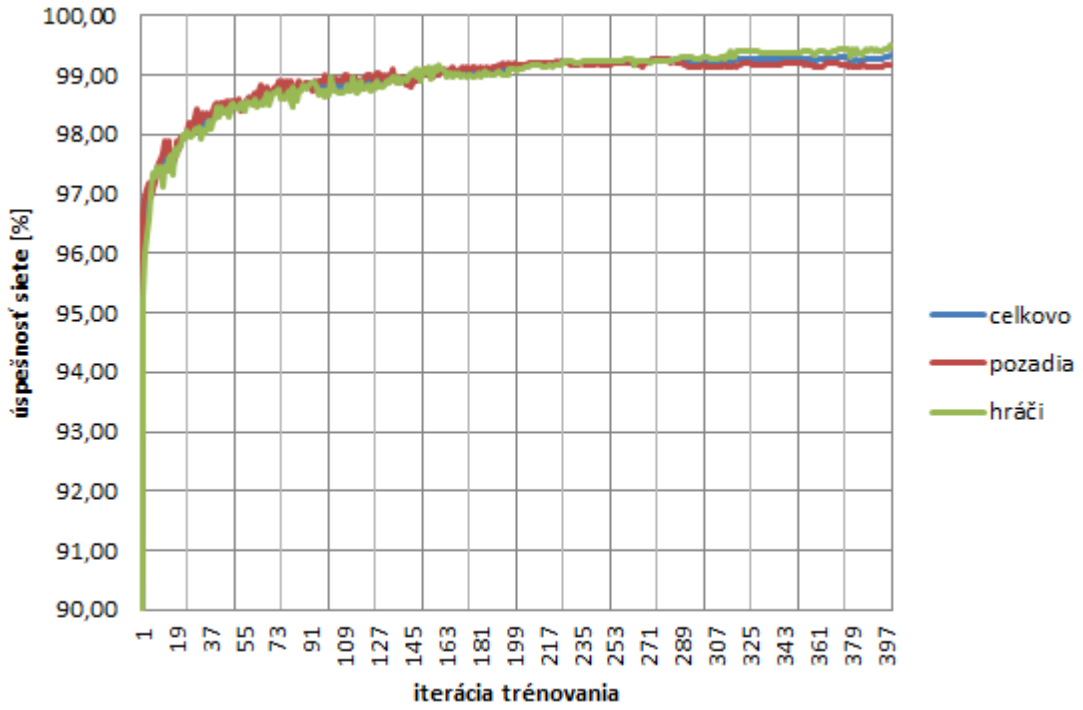
V predchádzajúcej kapitole sme navrhli konvolučnú neurónovú sieť, ktorú použijeme pre implementáciu detektora. Parametre tejto siete vidíme v tabuľke 6.1. V tejto kapitole podrobnejšie analyzujeme túto sieť. Zistíme, aké vzory pri testovaní robili sieti problémy, zobrazíme interné stavy siete a podrobnejšie otestujeme i odolnosť voči šumu.

6.1 Priebeh trénovania a testovanie

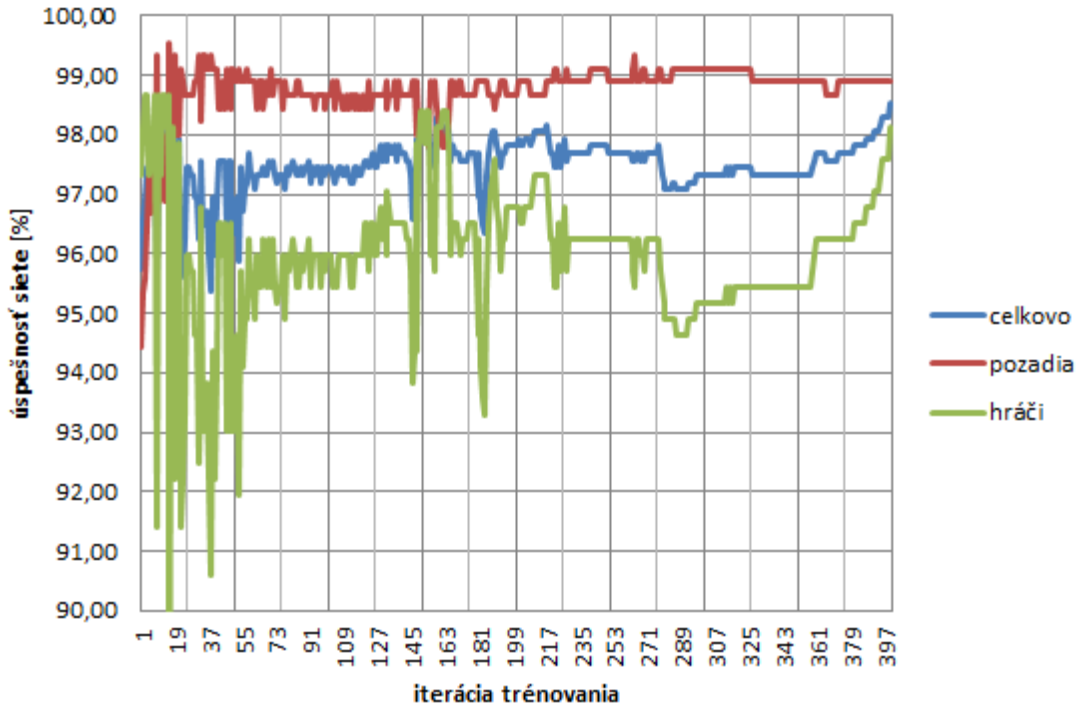
Priebeh učenia je možné vidieť na obrázku 6.1. Úspešnosť na testovacej množine počas trénovania zobrazuje obrázok 6.2. Učenie sme zastavili po 400 iteráciách. Z grafu môžeme vyčítať, že sa sieť naučila trénovacie vzory relatívne rýchlo (peknú úspešnosť máme už po dvestej iterácii), naopak testovaciu množinu vedela sieť rozpoznávať s malou chybou až ku koncu učenia. Tabuľka 6.2 zobrazuje úspešnosť siete pre jednotlivé triedy (pozadie a hráč) po natrénovaní siete. Obe triedy majú porovnateľnú úspešnosť.

typ vrstvy	počet máp	konvolučné jadro	veľkosť mapy
vstup	3		15×24 px
konvolučná	4	4×4	12×21
subsamplingová	4	3×3	4×7
konvolučná	32	3×3	2×5
subsamplingová	32	2×5	1×1
neurónová	50 neurónov		
výstup	2 neuróny		

Tabuľka 6.1: Parametre konvolučnej siete *Net1.6-15-4-32* s farebným vstupným obrázkom veľkosti 15×24 px.



Obr. 6.1: Priebeh trénovania siete *Net1.6-15-4-32*. Môžeme pozorovať, že chyba na trénovacích vzoroch rýchlo klesá už po pár trénovacích iteráciách.



Obr. 6.2: Priebeh chyby na testovacích vzoroch.

trieda	úspešnosť tren. množ.	úspešnosť test. množ.
hráč	99,51%	98,12%
pozadie	99,17%	98,89%

Tabuľka 6.2: Úspešnosť jednotlivých tried siete v *trénovacej* a *testovacej množine 1* po naučení siete.

Keďže chyba siete nebola nulová, pozrime sa, ktoré vzory boli pre sieť najväčším problémom pri učení a testovaní. Obrázky 6.3 a 6.4 zobrazujú zle klasifikované vzory z jednotlivých tried v trénovacej a testovacej množine. Vzorky hráčov, ktoré sieť označila za pozadie obsahovali najmä tieto situácie:

- hráč bol pri mantineli a splýval tak s pozadím
- hráč bol rozmazaný
- hráč vo vzore zaberal príliš málo miesta
- na snímke bolo hráčov viac

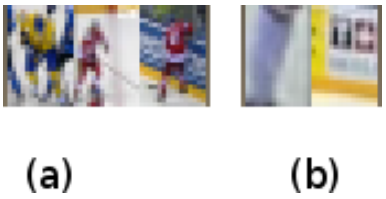
Naopak, pozadia, ktoré boli označené za hráčov, zväčša obsahovali nejaký farebný vzor uprostred bielej ľadovej plochy, prípadne nejakého hráča časťou zachytávali.

6.2 Odolnosť voči šumu

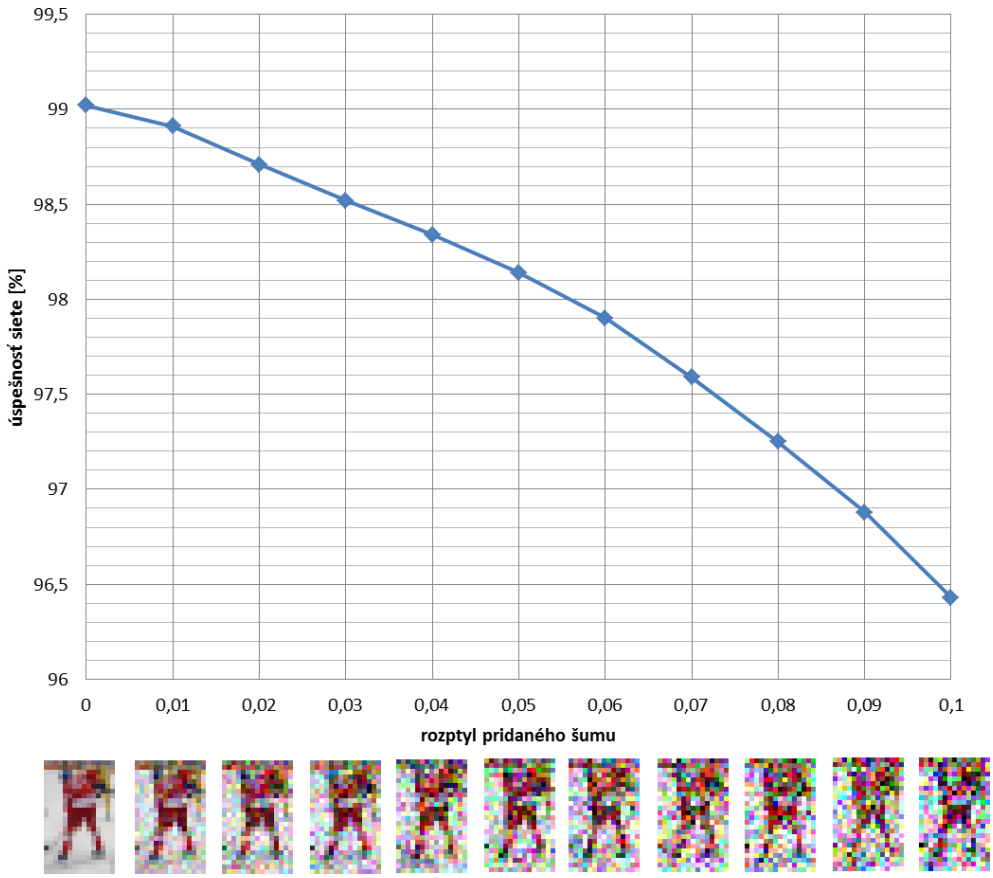
Pozrime sa bližšie, ako táto naučená sieť[1] zvláda rôzne úrovne pridaného Gaussovského šumu. Testovali sme na zašumenej *testovacej množine 1.* Test sme rozdelili na testovanie malého šumu (rozptyl od 0,01 do 0,1) a testovanie veľkého šumu (rozptyl 0,1 až 1). Pre každú hodnotu šumu sme otestovali danú, rôzne zašumenú množinu 100krát. Priemerné úspešnosti siete zobrazujú grafy na obrázkoch 6.5 a 6.6. Pri šume do rozptylu 0,1 si sieť drží peknú úspešnosť, potom ale úspešnosť na viac zašumených vzoroch rýchlo klesá. Je však až prekvapujúce, že sieť správne zaradila v priemere až 75% vzorov pri šume s rozptylom 1. Pri tomto šume nebol totiž pôvodný obraz voľným okom rozpoznateľný. Odolnosť siete voči šumu považujeme za obzvlášť dobrú.

[1] sieť nebola učená pomocou zašumených vzorov

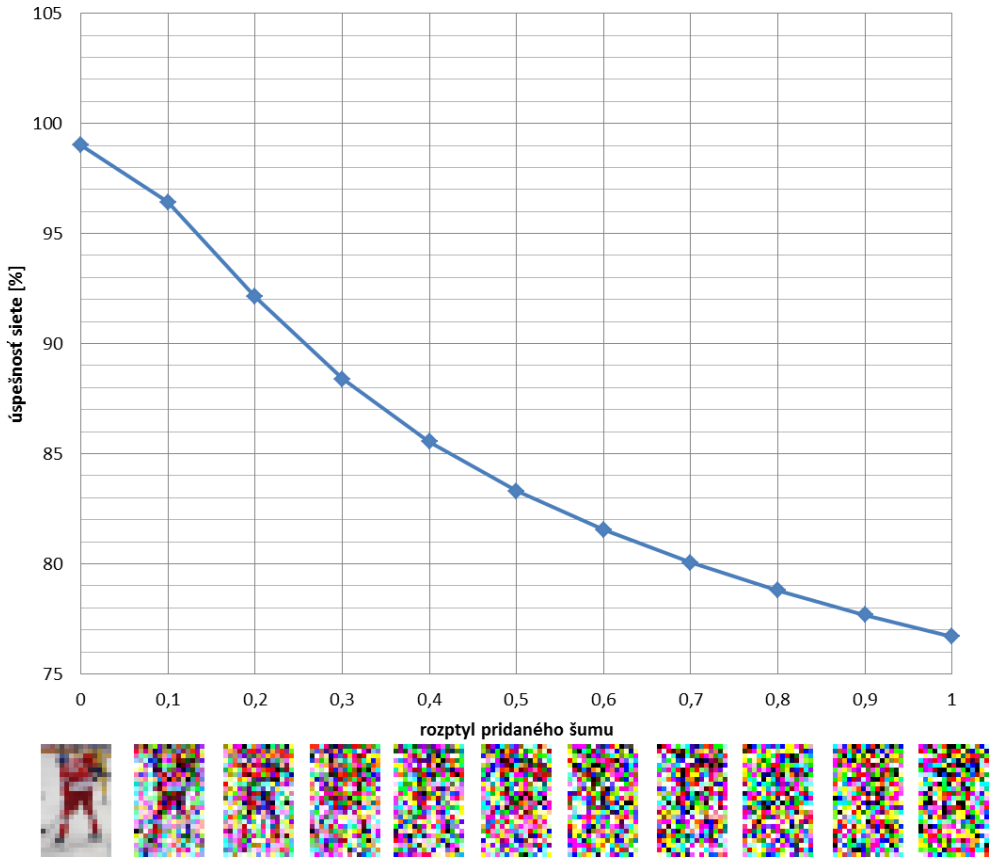
Obr. 6.3: Nesprávne klasifikované vzory z trénovacej množiny. Podobrázok (a) zobrazuje vzory z triedy *hráč* ktoré boli klasifikované ako trieda *pozadie*. Podobrázok (b) zobrazuje opačnú situáciu. Najväčšie problémy robili sieti vzory hráčov, ktoré neboli presne vycentrované, alebo hráči, ktorý stáli pri mantineli.



Obr. 6.4: Nesprávne klasifikované vzory z testovacej množiny. Podobrázok (a) zobrazuje vzory z triedy *hráč* ktoré boli klasifikované ako trieda *pozadie*. Podobrázok (b) zobrazuje opačnú situáciu. Problém robili sieti podobné prípady ako vzory z trénovacej množiny.



Obr. 6.5: Odolnosť natrénovanej siete voči šumu na zašumenej testovacej množine 1. Hodnoty sú priemer zo 100 rôznych behov.



Obr. 6.6: Odolnosť natrénovanej siete voči šumu na zašumenej testovacej množine 1. Hodnoty sú priemer zo 100 rôznych behov.

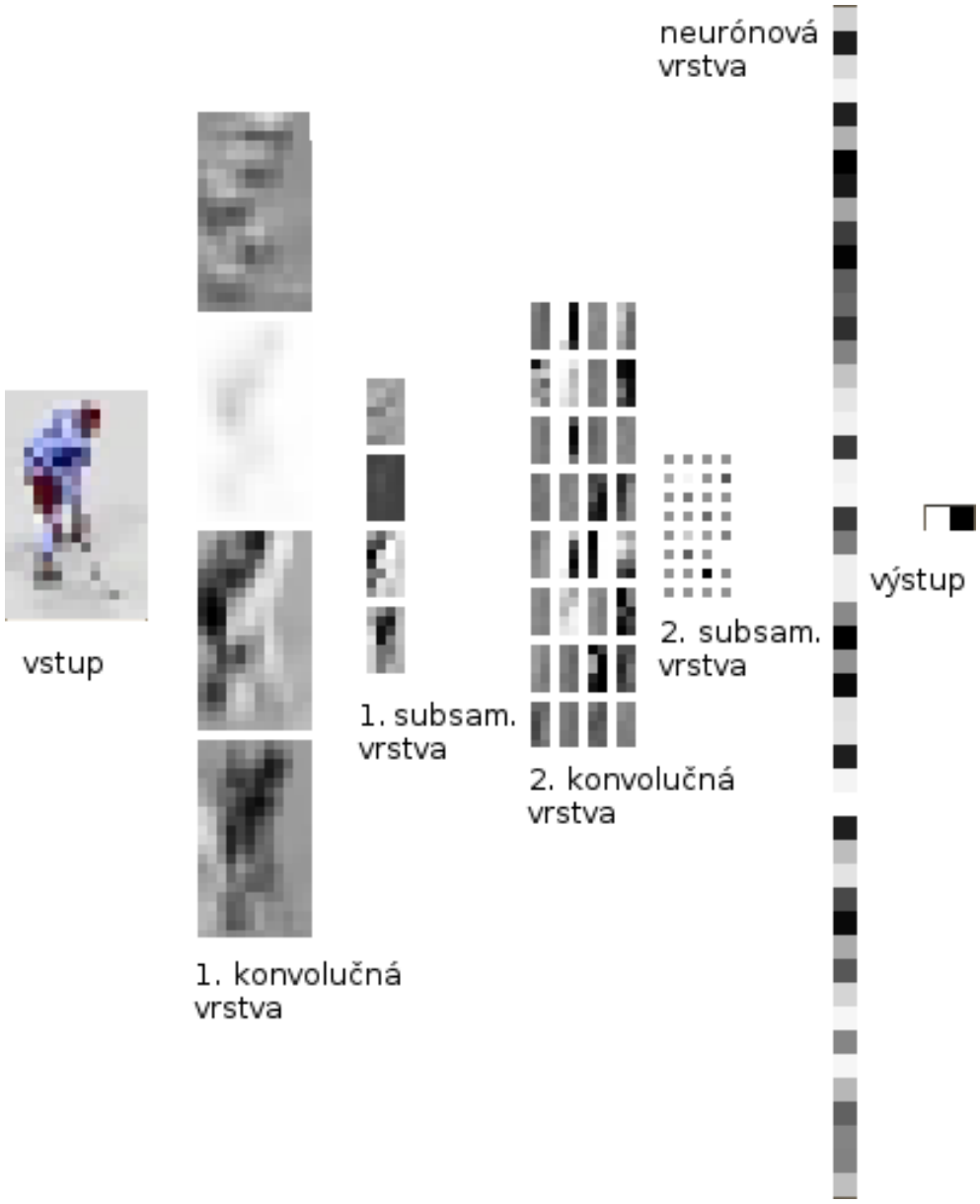
6.3 Interné stavy siete

V konvolučných sietiach môžeme pozorovať aktivity v jednotlivých príznakových mapách v grafickej podobe. Sú to totiž dvojrozmerné matice, ktorých hodnoty sa vzťahujú priamo k vstupnému obrazu. Sú výsledkom konvolučných operácii na obraze v predchádzajúcej vrstve. Môžeme tak vidieť (aspoň v prvej vrstve), aké príznaky sa jednotlivé príznakové mapy naučili, teda príslušné neuróny budú aktívne, detekovať. V ďalších vrstvách toto pozorovanie už nie je také jednoduché, pretože mapy sú menšie, je ich viac a ich hodnoty už nie sú kombinácie priamo z obrazu. Pozrime sa na interné stavy pre hráčov (obrázky 6.7 a 6.8) a interné stavy pozadí (obrázky 6.9 a 6.10). Obrázky zobrazujú príznakové mapy pre jednotlivé vrstvy siete. Biely pixel v mape predstavuje neurón, ktorý je plne aktívny, čierny pixel predstavuje neurón bez aktivity. Prvá konvolučná vrstva obsahuje štyri príznakové mapy. Môžeme pozorovať, že tvoria akési hranové detektory. Prvá je aktívna najmä na horizontálnych čiarach, druhá na plochách bez štruktúry, tretia na pravých hranách objektov a štvrtá na ľavých hranách objektov. Samozrejme, že tieto príznakové mapy detekujú i nejaké iné príznaky, ktoré však pohľadom odhadnúť nevieme. Ďalšie vrstvy v sieti detekujú zložitejšie príznaky, ktoré sú kombináciami príznakov z predchádzajúcich vrstiev. Môžeme pozorovať, že v poslednej subsamplingovej a neurónovej vrstve sú pre rôznych hráčov korešpondujúce hodnoty výstupov neurónov podobné. Podobne to platí i pre rôzne pozadia. To znamená, že hlavná časť rozpoznávania prebehla práve v konvolučných a subsamplingových vrstvách.

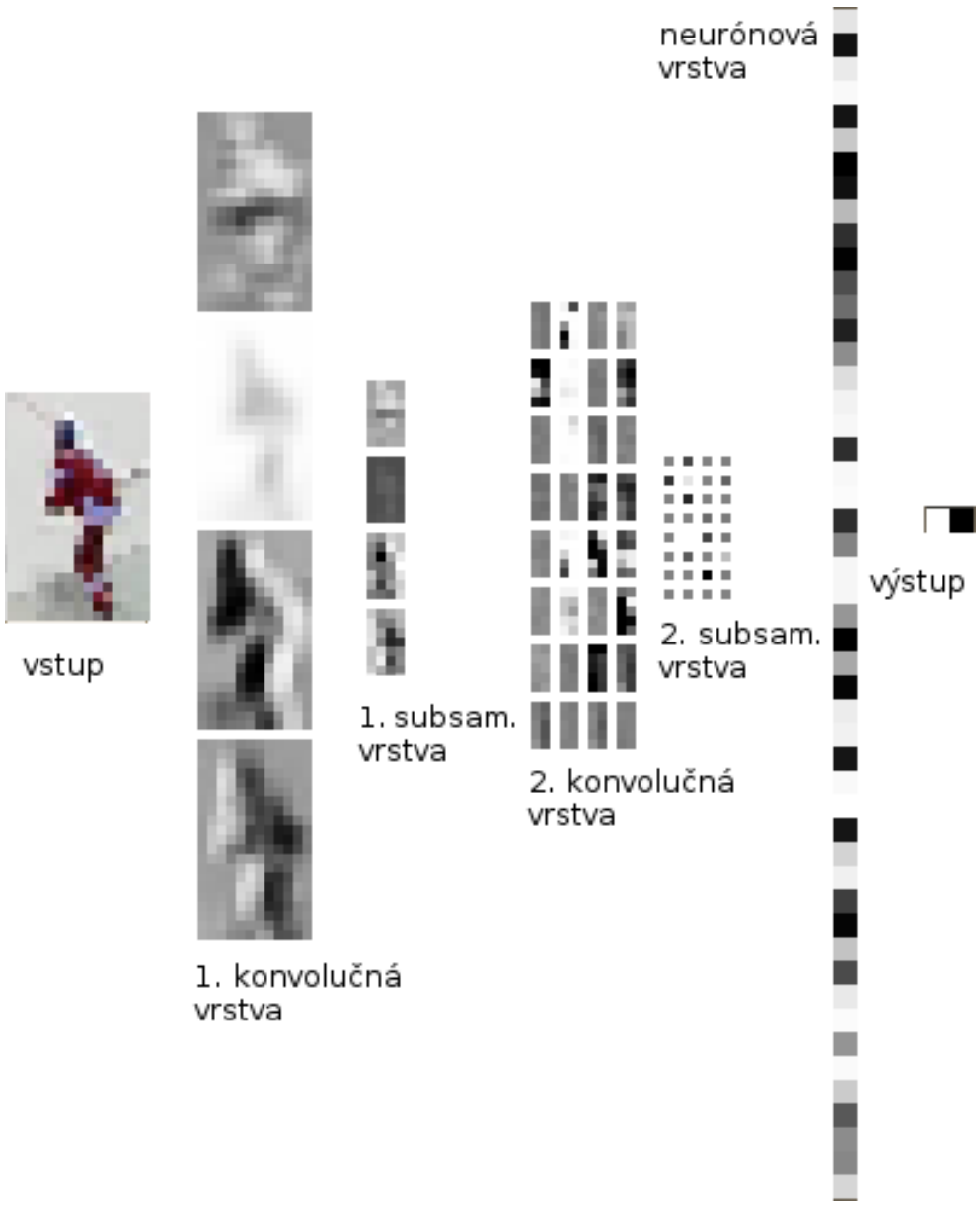
Na obrázku 6.11 vidno stavy siete, ktorej bol predložený zašumený vzor. Už príznakové mapy v prvej konvolučnej vrstve majú podobné výstupy ako v prípade nezašumeného obrázku. V ďalších vrstvách je podobných výstupov oveľa viac. Sieť dokáže interne šum potlačiť. To je pravdepodobne možné najmä vďaka subsamplingovým vrstvám, ktoré zjednodušene povedané zmenšia a spriemerujú vstup, čím klesá hodnota šumu (viac o potlačovaní šumu sme napísali v kapitole 4).

6.4 Odolnosť voči posunu a zmene mierky

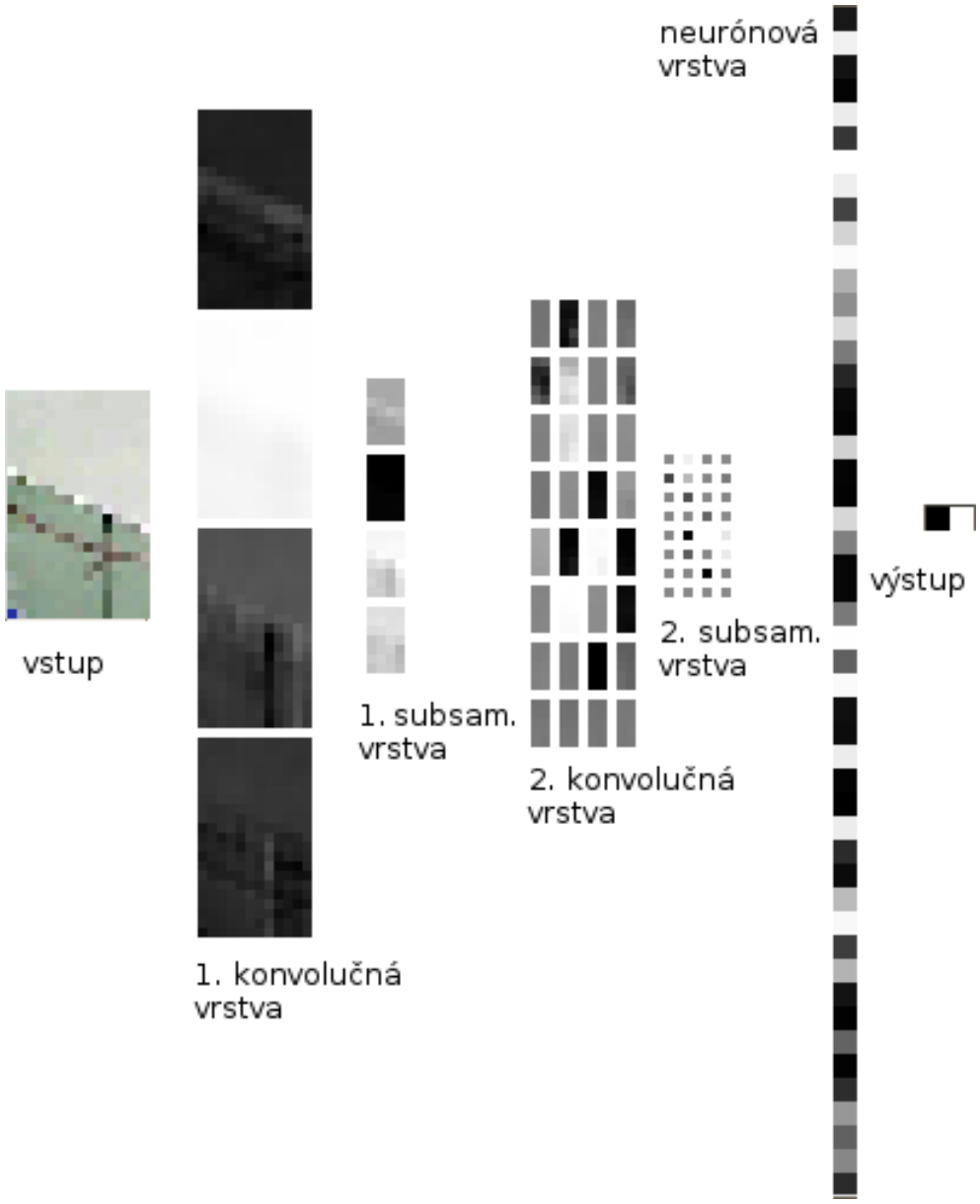
Ako sme už v tejto práci písali mnohokrát, konvolučné siete by mali byť odolné voči menším geometrickým transformáciám, ako je napríklad posun, či zmena mierky. Teda v našom konkrétnom prípade by mali detekovať hokejistu, i keď nebude priamo v strede skúmaného výrezu. Pozorovať to môžeme na obrázku 6.12. Jednotlivé farebné obdĺžniky predstavujú výrezy, na ktorých nám sieť detekovala hráča. Vidíme, že sieť je odolná voči posunu ba dokonca hráča detekuje i keď je v predloženom výreze iba jeho časť. Túto vlastnosť využijeme pri implementácii detektora, kde môžeme detekčným výrezom posúvať po väčších krokoch a nemusíme použiť viac jeho veľkostí.



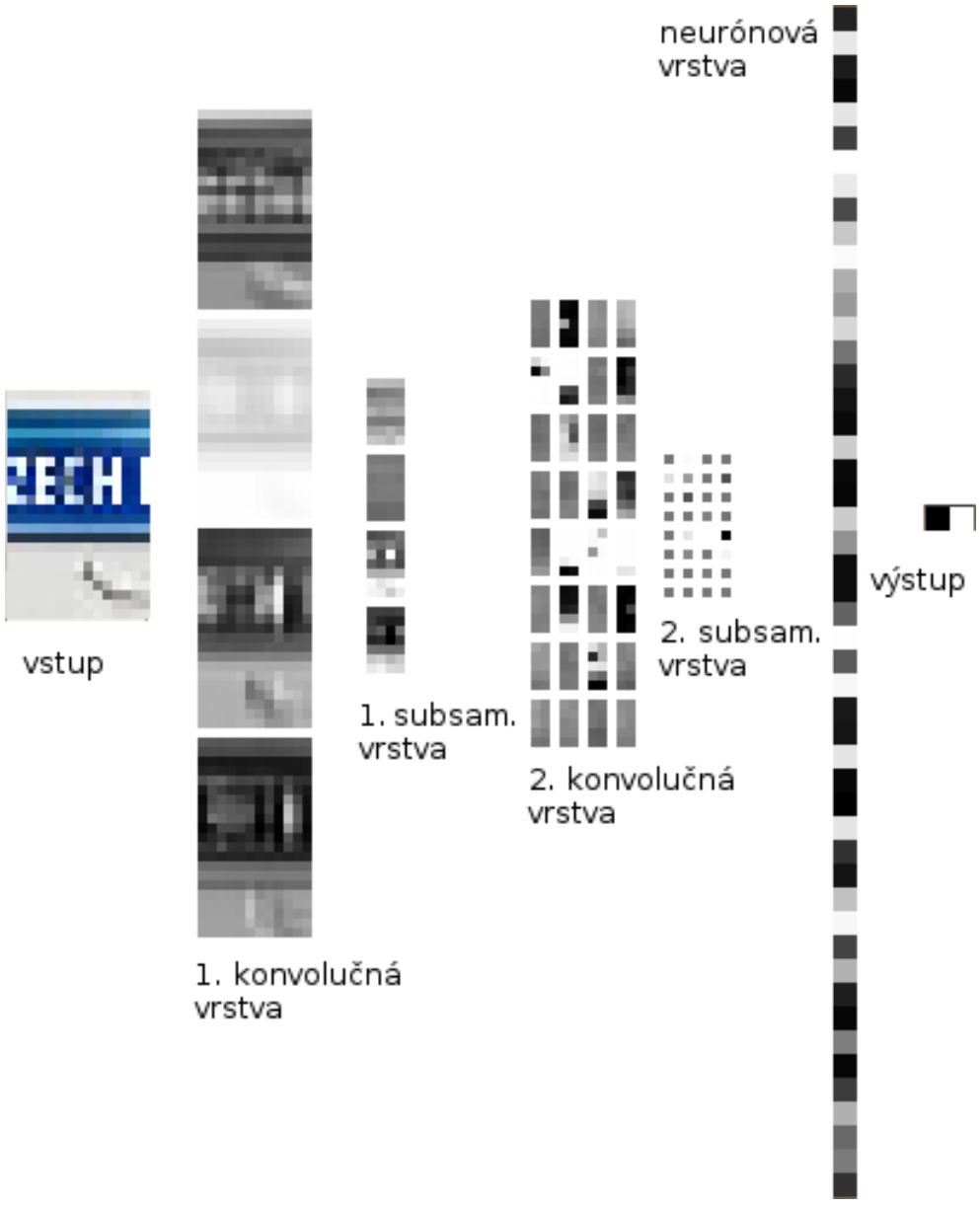
Obr. 6.7: Interné stavy naučenej siete - príznakové mapy pri predložení hráča na vstup



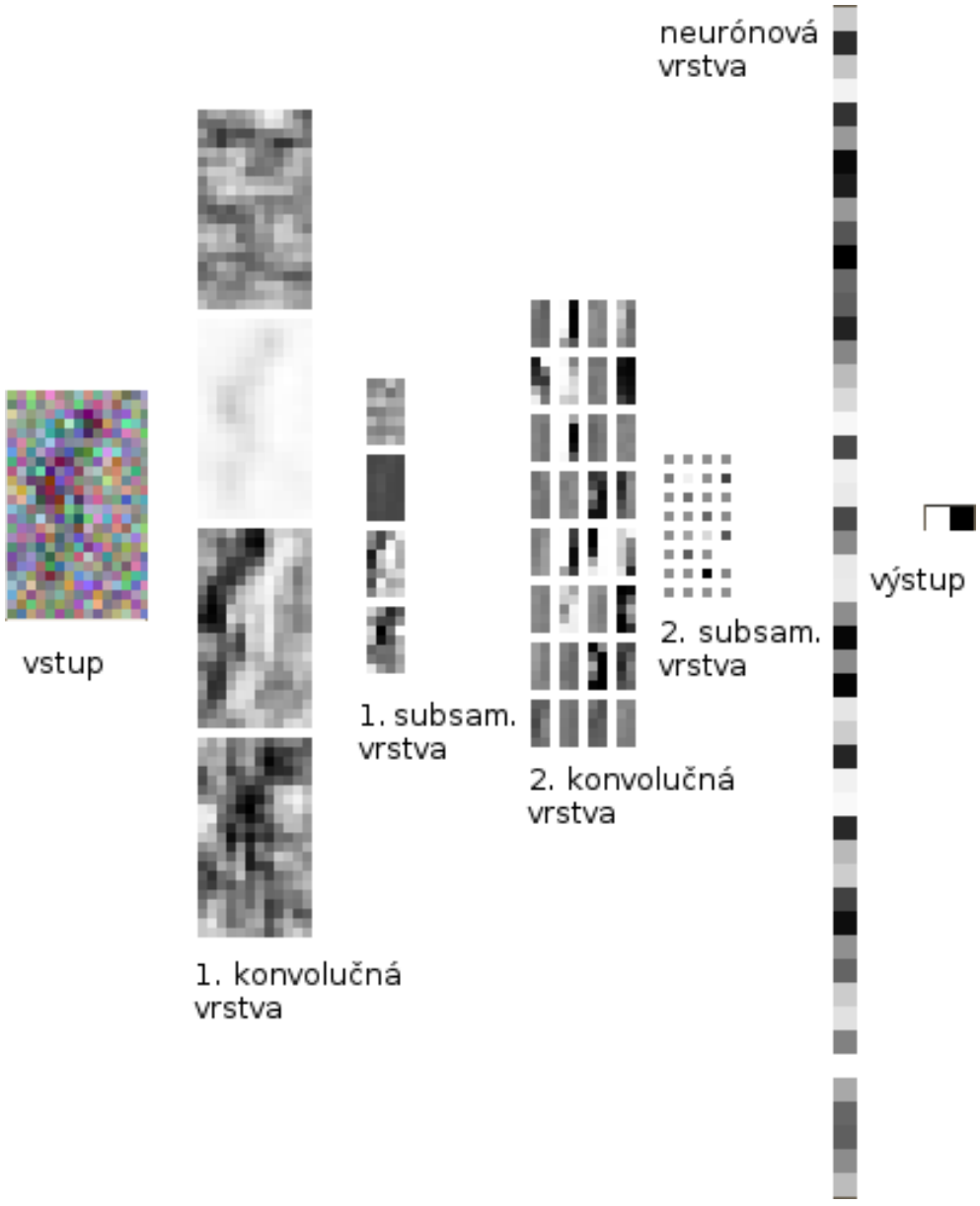
Obr. 6.8: Interné stavy naučenej siete - príznakové mapy pri predložení hráča na vstup



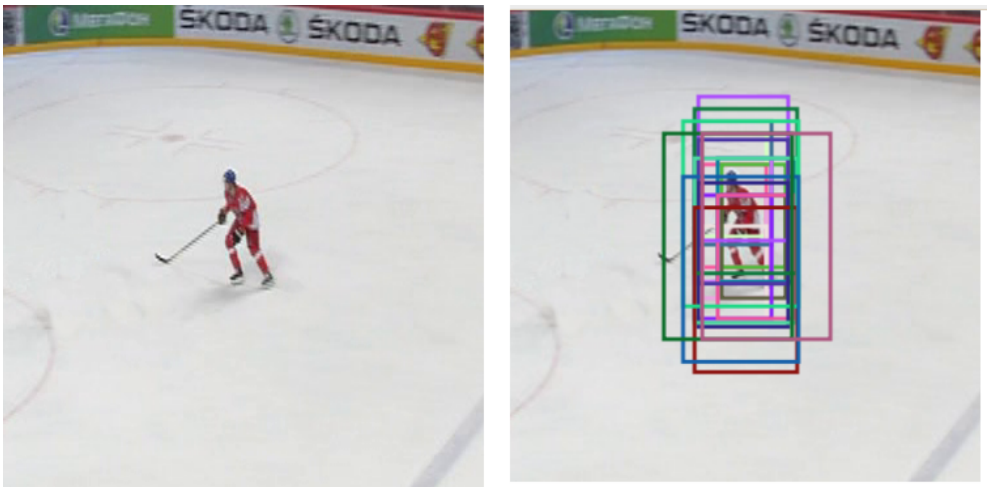
Obr. 6.9: Interné stavy naučenej siete - príznakové mapy pri predložení pozadia na vstup



Obr. 6.10: Interné stavy naučenej siete - príznakové mapy pri predložení pozadia na vstup



Obr. 6.11: Interné stavy naučenej siete - príznakové mapy pri predložení zašumeného hráča na vstup



Obr. 6.12: Odolnosť siete voči posunu a zmene veľkosti vzoru. Naľavo vidíme pôvodný vzor - hokejistu, napravo farebné obdĺžniky rôznych veľkostí a rôzneho umiestnenia, kde sieť hokejistu detekovala.

Kapitola 7

Možnosti rozšírenia detektora

V predchádzajúcej kapitole sme navrhli sieť, ktorá spoľahlivo rozpoznáva hokejového hráča od pozadia. K detekovanému hokejistovi môžeme zobraziť rôzne informácie. Bohužiaľ rozpoznanie konkrétneho hokejistu je z jedného obrázka takmer nemožné, pretože tvár je príliš malá a číslo na hokejistovi nie je vždy a spoľahlivo vidno. V našej práci sa pokúsime aspoň každého hokejistu priradiť do svojho tímu.

7.1 Zaradenie hráča do tímu

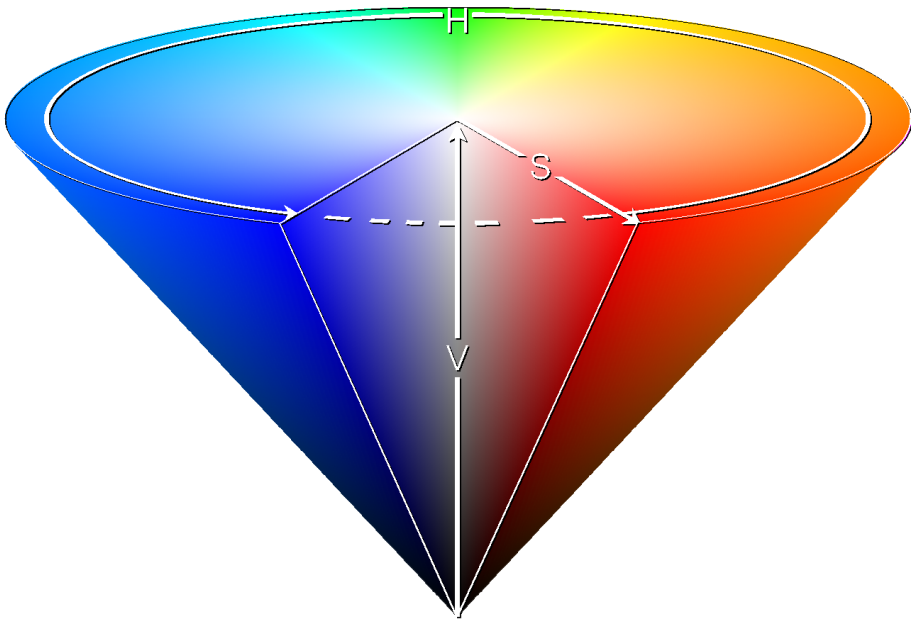
Tímy sa v zápasoch od seba odlišujú najmä farbou dresu, skúsime teda z hráča extrahovať hlavnú farbu dresu a potom ho automaticky priradiť k tímu. Hokejisti rovnakých alebo podobných farieb budú patriť k sebe. Budeme postupovať takto:

- na hracej ploche nájdeme hokejistov
- každému z nich priradíme jeho farbu
- vytvoríme dva zhluky, každý pre jeden tím
- hráčov priradíme do tímu, ktorý reprezentuje jeho zhluk

Prvý bod za nás robí detektor. Ďalšie kroky si teraz podrobnejšie rozpíšeme.

Priradenie farby k hráčovi

Každý hokejový hráč je počas hry oblečený do dresu, ktorého farba reprezentuje jeho tím. Detektor hráčov nám hráča nájde a označí. Hráč bude v ideálnom prípade umiestnený približne v strede tohto označenia. V jeho okolí bude zväčša biela ľadová plocha. Každému hráčovi priradíme farbu, ktorá bude priemer farieb v strede označenia. Ak bude hráč červený bude táto farba približne červená a pod.



Obr. 7.1: Znázornenie farebného priestoru definovaného pomocou *HSV*. Zdroj obrázku *wikipedia*.

Vytvorenie zhlukov

Farba môže byť v počítačovom svete reprezentovaná rôzne. Najčastejšie sa zapisuje pomocou modelu *RGB*, kde sa farba skladá z troch farebných zložiek, a to *červenej*, *zelenej* a *modrej*. Pomocou kombinácie týchto troch farieb dostávame takmer každú farbu. Tento model však nie je dobrý pre skúmanie podobnosti farieb, pretože podobné farby (z pohľadu človeka) nemusia ležať v tomto priestore blízko seba. Ďalší hojne používaný model pre reprezentáciu farieb *HSV* už použiť môžeme. Ten definuje farby na základe odtieňu, saturácie a hodnoty. Farby s podobným odtieňom ležia vedľa seba. Grafické znázornenie tohto farebného modelu vidíme na obrázku 7.1. Farbu, ktorú sme priradili hráčovi, prevedieme do priestoru *HSV*, potom každú z nich priradíme do jedného z dvoch zhlukov (každý za jeden tím). Ako zhlukovací algoritmus použijeme známy algoritmus *K-means* a ako funkciu pre vzdialenosť euklidovskú vzdialenosť medzi vektormi, ktoré reprezentujú farbu v *HSV* priestore.

K-means

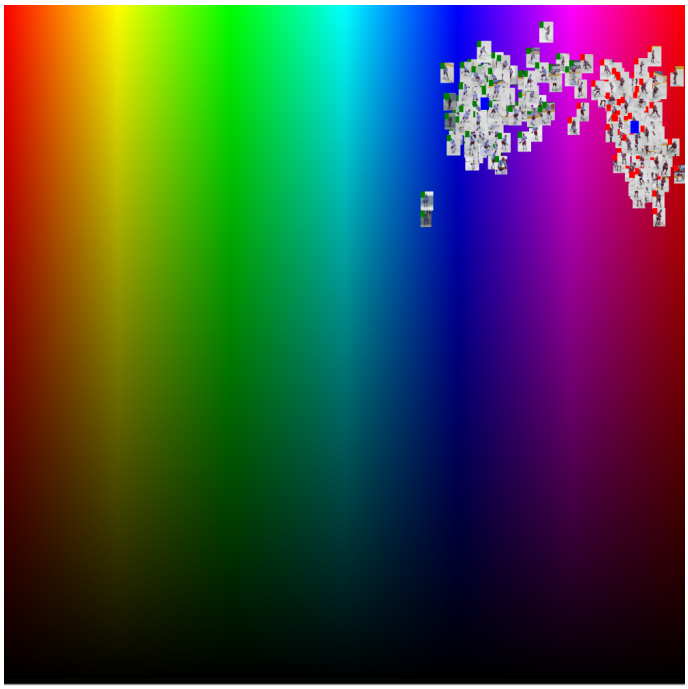
Pre zaradenie farby do zhluku použijeme algoritmus *K-means*. Tento algoritmus priradí vzor do toho zhluku, ku ktorému je vzor bližšie. Zhluk je reprezentovaný svojím stredom. Algoritmus hľadá pre množinu vstupných dát $X = \{x_1, x_2, \ldots, x_n\}$ vektory $m_1, \ldots, m_k$, kde k je počet hľadaných zhlukov, také, ktoré minimalizujú strednú

kvadratickú odchýlku vzorov z X od vektorov m. Počet zhlukov musí byť vopred určený. V našom prípade sú to zhluky dva. Presný algoritmus vidíme zapísaný algoritmom 7.1.

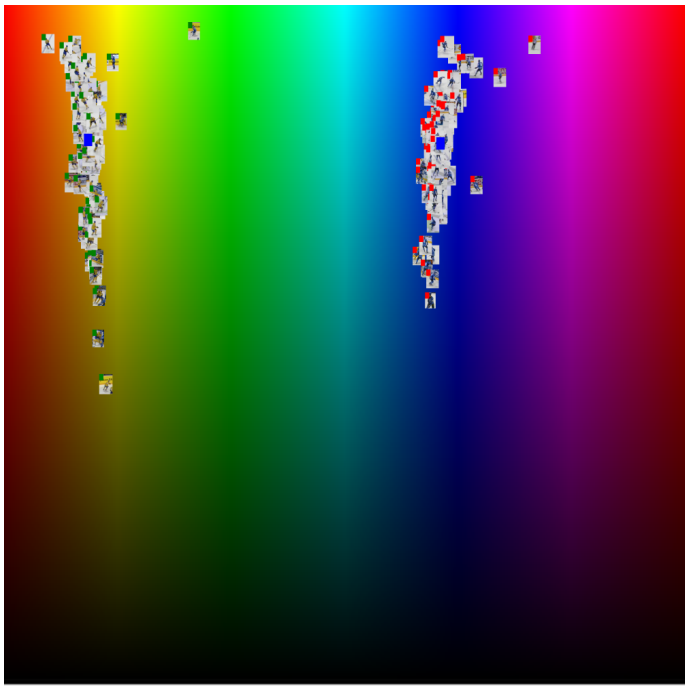
Algorithm 7.1 Algoritmus učenia vrstevnatej neurónovej siete

$X \leftarrow \{x_1, x_2, \ldots, x_n\}$ {Množina vstupných vzorov.}
$k \leftarrow 2$ {Počet zhlukov, v našom prípade dva.}
{Inicializácia stredov zhlukov, zväčša ako náhodný vzor z X.}
for $j = 1 \rightarrow k$ **do**
  $m_j \leftarrow$ náhodný vzor z X
end for
repeat
  {Klasifikácia}
  for all $x_i, i = 1 \ldots n$ **do**
    {Priradenie vzoru x_i do triedy y_i}
    $y_i \leftarrow \arg\min_{j=1\ldots k} \|x_i - m_j\|$
  end for
  {Adaptácia}
  for $j = 1 \rightarrow k$ **do**
    {Určenie nového stredu zhluku m_j}
    $S_j \leftarrow \{x_i : y_i = j; i = 1 \ldots n\}$
    $m_j \leftarrow \frac{1}{|S_j|} \sum_{x_i \in S_j} x_i$
  end for
until nebude splnené ukončovacie kritérium
{ukončovacie kritérium je, keď žiaden zo vzorov sa nepresunie do inej triedy ako bol klasifikovaný v predchádzajúcej iterácii.}

Pomocou tohto algoritmu priradíme každému hráčovi jeden z tímov. Ako takéto zhluky vyzerajú vidíme na obrázkoch 7.2a a 7.2b. Detail zhluku zobrazuje obrázok 7.3, je na ňom vidno, že väčšina hokejistov je správne zaradená. Táto metóda bude dobre fungovať v prípade dobre farebne oddelených dresov. V prípade podobných farieb by bolo vhodné sa zamerať na iné rozlišovacie znaky na dresoch jednotlivých hráčov, to je ale už nad rámec tejto práce. Uvedenú metódu rozhodovania, do ktorého tímu patrí, sme navrhli a implementovali najmä ako ukážku možností ďalšieho rozširovania navrhnutého detektora.

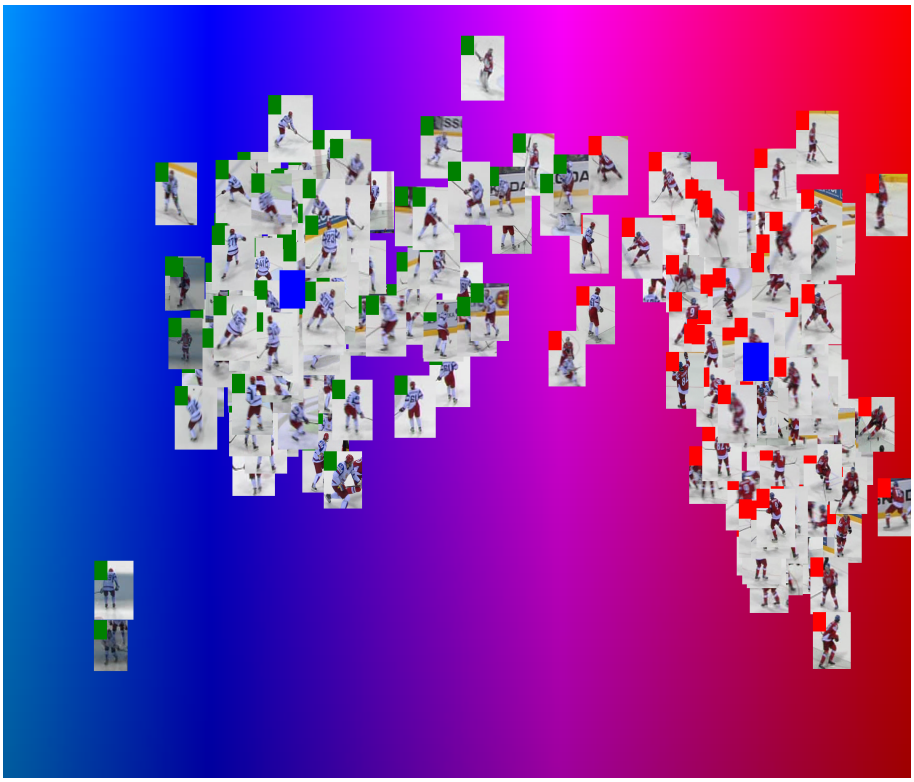


(a) Zápas Česko – Rusko.



(b) Zápas Švédsko – Fínsko.

Obr. 7.2: Zobrazenie hráčov ako zhluky podľa ich priemernej farby vo farebnom priestore *HSV*. Použité boli iba hodnoty *odtieňa* a *saturácie*. Každý zhluk reprezentuje jeden tím.



Obr. 7.3: Detail zhlukov v zápas Česko – Rusko. Hráči z jedného tímu sú označení zeleným štvorčekom v rohu, hráči druhého červeným. Modrými štvorčekmi sú vyznačené stredy zhlukov.

Kapitola 8

Výber frameworku

Pre implementáciu konvolučných neurónových sietí sme sa rozhodli vybrať nejakú už existujúcu knižnicu – framework, najmä kvôli rýchlosti. Hlavnou požiadavkou bolo, aby efektívne implementoval konvolučné siete, ďalej sme zohľadňovali rýchlosť učenia a rozpoznávania. Hodnotili sme i možnosti použitia a dostupnú dokumentáciu. Do užšieho výberu sme vybrali dve knižnice, prvou z nich je *EBLearn - Energy Based Learning*, druhou *Torch7* [3]. Obe sú distribuované pod *BSD licenciou*, takže nie je problém ich použiť v akomkoľvek projekte. Obe knižnice boli v dobe písania tejto práce aktívne vyvíjané. V nasledujúcom texte si popíšeme a porovnáme jednotlivé knižnice a nakoniec jednu z nich vyberieme pre implementovanie nášho riešenia.

8.1 EBLearn

Knižnicu sme našli priamo na stránke jedného z autorov konvolučných sieti Yann LeCuna, ktorý sa priamo podieľa na jej vývoji. To by malo zaručovať korektnosť implementácie. Je to multiplatformová open-source knižnica pre strojové učenie napísaná v C++ šírená pod BSD licenciou. Bola navrhnutá pre vytváranie a trénovanie systémov pre klasifikačné, regresné a detekčné úlohy. Poskytuje API priamo pre programovanie v C++. Jej hlavnou výhodou je, že obsahuje nástroje obsluhované konfiguračnými súbormi bez nutnosti siete priamo programovať v C++. Nevýhodou je, že *EBLearn* nie je veľmi kvalitne dokumentovaná a v priebehu vývoja sa niekoľkokrát zmenil formát konfiguračných súborov. V neposlednom rade je niekedy dosť nestabilná, učenie neurónovej siete niekedy skončilo chybou segmentation fault. I napriek týmto nevýhodám sa jedná o komplexný nástroj, ktorý umožňuje prácu s optimalizovanými algoritmami. S knižnicou sa dá pracovať priamo v jazyku C++, ale sami autori odporúčajú používať pripravené nástroje. Popíšeme použitie niektorých z nich.

Príprava dát

Pre prípravu dát slúži program *DsCompile*. Tento nástroj načíta vstupné obrázky a prevedie ich do binárneho formátu, ktorý sú schopné spracovať ďalšie nástroje. Dáta, v našom prípade obrázky, dokáže zmenšiť na požadovanú veľkosť, odobrať farbu, prípadne dokáže obrázky deformovať. Deformácie môžu byť posunom alebo otočením. Pretože sa pri deformáciách strácajú informácie z okrajových častí, môžu byť tieto časti doplnené obrázkom pozadia. Použitie tohto nástroja je nasledujúce:

```
dscompile samples/ -dims 15x27x3 -outdir samples-out -precision double
```

Uvedený príklad načíta vstupný priečinok samples/ ktorý obsahuje vzorky pre jednotlivé triedy v podpriečinkoch. Pomocou parametra dims nastavíme výstupnú veľkosť vzorov. Výstup sa zapíše do adresára samples-out. Presnosť môžeme nastaviť prepínačom -precision.

Ďalšie nástroje *DsSplit* a *DsMerge* pracujú už s formátom dát, ktorý sme dostali po použití predchádzajúceho nástroja. Prvý z nich dokáže rozdeliť množinu dát, ktorú sme dostali použitím predchádzajúceho nástroja. Druhý dokáže spojiť dve rôzne množiny. My sme ich použili na vytvorenie trénovacej, testovacej a validačnej množiny. Použitie nástroja *DsMerge* je nasledujúce:

```
dsmerge root ds0 ds1 ds2
```

Uvedený príklad spojí množiny vzorov ds0 a ds1 do množiny ds2, množiny sa nachádzajú v priečinku root.

Vstupy pre tieto nástroje sa predávajú pomocou prepínačov pri volaní jednotlivých programov.

Trénovanie a testovanie

Asi najdôležitejší nástroj knižnice predstavuje nástroj *Train*. Umožňuje vytvoriť konvolučnú sieť zadaných parametrov a túto sieť, podľa zadaných parametrov natrénovať. Sieť sa vytvára v konfiguračnom súbore, kde sa definujú jednotlivé vrstvy a ich parametre. Formát týchto súborov môžeme vyčítať z príkladov priložených ku knižnici. Ako vstup môže dostať trénovaciu a validačnú množinu. V prípade, že chceme iba testovať, použije sa množina zadaná ako trénovacia iba na testovanie. Je možné nastaviť rôzne parametre pre trénovanie. Nevýhodu vidíme vo výstupe z tohoto nástroja, ktorý je iba textový výstup do konzole (obsahuje informácie o priebehu trénovania ako aj výsledky/chybovosť po jednotlivých iteráciách). Formát nie je možné meniť. Pre spracovanie týchto údajov, napríklad v podobe tabuľky alebo grafu, je nutné napísať si vlastný parser, ktorý výstup spracuje.

Príklad konfiguračného súboru pre nástroj train, zobrazujeme časť, kde sa vytvára sieť:

```
#layers
c0=conv0
s1=subs1
c2=conv2
s3=subs3
f4=linear4
f5=linear5

arch=${c0},${s1},${c2},${s3},${f4},${f5}

inputh=23
inputw=15

conv0_kernel=4x4
conv0_stride=1x1
conv0_table_in=1
conv0_table_out=8
subs1_kernel=2x2
subs1_stride=2x2
...
```

Sieť natrénujeme predaním tohoto súboru utilite *Train*:

```
train sample.conf
```

Knižnica *Eblearn* obsahuje i nástroj *Detect*. Program dokáže načítať naučenú neurónovú sieť a vstupné obrázky v rôznych formátoch (adresár, jednotlivé obrázky, video, webkamera). V týchto obrázkoch hľadá naučené vzory. Pokiaľ je to možné, využíva pri hľadaní viac vlákien, čo urýchľuje dosiahnutie výsledku. Tento nástroj sa nám ale na našej sieti nepodarilo rozchodiť (chyba floating point exception) i keď na priloženom príklade s rozpoznávaním tváre fungoval bezchybne. Pre detekciu by sme si teda museli napísať vlastný nástroj.

Optimalizácie

V dobe písania tejto práce umožňovala aktuálna verzia *Eblearn* 1.1 využiť rôzne optimalizácie pre urýchlenie výpočtu. Podľa stránok projektu sú to optimalizácie využívajúce tieto knižnice:

- TH: (TH Tensor Library, ktorý je fork časti knižnice Torch-7) autori deklarujú zrýchlenie od 30% do 100% pre trénovanie aj detekciu.
- IPP: až 100% zrýchlenie detekcie.
- OpenMP: 0 až n-krát rýchlejšie, kde n je počet dostupných jadier. Zatiaľ je podpora tejto knižnice v *Eblearn* iba experimentálna.
- GPU: podpora výpočtu na grafickej karte je zatiaľ neimplementovaná, ale autori sľubujú skoré implementovanie.

Pre testovanie rýchlosti použijeme optimalizácie za pomocí knižníc TH a OpenMP, ktoré sme mali k dispozícii.

Zhrnutie

Eblearn sa ukázala ako veľmi dobrá knižnica na prácu s konvolučnými neurónovými sieťami. Používateľsky prívetivé, i keď nie veľmi dobre dokumentované konfiguračné súbory, nám umožňujú vytvoriť ľubovoľnú konvolučnú neurónovú sieť. Táto sieť sa natrénuje a otestuje. Komplexný program, ktorý by obsahoval prípravu dát, trénovanie, testovanie i vyhodnotenie výsledkov môže byť napísaný v akomkoľvek jazyku, ktorý umožňuje volanie príkazov z príkazového riadku. Nám sa osvedčil programovací jazyk *Bash*. Ako klad tiež hodnotíme komunikáciu s autormi projektu pri riešení niektorých problémov, s ktorými sme sa stretli pri používaní tejto knižnice. Bohužiaľ, problém s nástrojom *detect* sa nám v čase písania práce nepodarilo s autormi knižnice vyriešiť.

8.2 Torch 7

Podobne ako *Eblearn* je *Torch 7* knižnica určená na strojové učenie. Je implementovaná v jazyku C. Rozhrania sprístupňuje pomocou skriptovacieho jazyku Lua. Rovnako ako *Eblearn* je i táto knižnica multiplatformová a šírená pod opensource BSD licenciou. Do základného balíka sa dajú doinštalovať ďalšie knižnice – balíčky, ktoré rozširujú jej funkčnosť, napríklad o prácu s obrázkami videom, kamerou, alebo s príkazovým riadkom. Pomocou jednoduchého skriptovania poskytuje, na rozdiel od *Eblearn*, i nízkoúrovňovú prácu s konvolučnými sieťami. Je tak možné si napríklad prispôsobiť výstup učenia a rovno pomocou jazyka Lua a pomocných knižníc ho spracovať do grafov a tabuliek.

Príprava dát

Pre prípravu dát neexistuje v základnej knižnici žiadna komplexná funkcia podobná nástroju *DsCompile*. Nie je však problém spracovať obrázky pomocou funkcií pre

prácu so súborovým systémom a následne použiť balíček *image* určený pre prácu s obrázkami. Ďalšou možnosťou je experimentálny balíček pre strojové učenie, balíček *nnx*, ktorý pridáva aj funkcie pre prípravu dát. I keď sa jedná o experimentálne rozšírenie, žiadne problémy s týmto balíčkom sme nezaznamenali. Riešenie dovoľuje jednou funkciou spracovať adresár s obrázkami. Funkcia upraví veľkosť obrázkov, prípadne odstráni farbu a pripraví dáta do podoby vhodnej pre ďalšie spracovanie. Tieto dáta uloží do cache, takže pri ďalšom spustení programu už nie je potrebné načítavať a konvertovať obrázky.

Trénovanie a testovanie

Neurónová sieť sa v *Torch 7* vytvára pomocou modulov. Moduly zapájame postupne za sebou (v prípade vrstevnatej siete). Môžeme ale vytvoriť i oveľa zložitejšie siete. Po vytvorení siete nastavíme kritérium, napríklad *MSE - stredná kvadratická chyba*, nastavíme parametre pre ukončenie učenia a pomocou jednej funkcie *train* sieť naučíme. Výstup v tomto prípade ovplyvniť nemôžeme. *Torch 7* poskytuje i nízkoúrovňovejšie učenie siete, a to tak, že vytvoríme cyklus, v ktorom pomocou pripravených funkcií spočítame gradient a upravíme váhy. Prípadne máme voľnosť, aké príkazy budeme ešte počas učenia vykonávať (napríklad zobrazovanie interných stavov siete, kreslenie grafov a pod.).

Príklad definovania a trénovania jednoduchej neurónovej siete v jazyku Lua pomocou knižnice *Torch 7*.

```
mlp = nn.Sequential()
mlp:add(nn.Linear(10, 25)) -- 10 vstupov, 25 v˜skrytej vrstve
mlp:add(nn.Tanh()) -- prenosová funkcia
mlp:add(nn.Linear(25, 1)) -- 1 výstup
criterion = nn.MSECriterion() -- kritérium je stredná kvadratická chyba
trainer = nn.StochasticGradient(mlp, criterion)
trainer:train(dataset) -- trénovanie siete na vzoroch z˜poľa dataset
```

Torch 7 neobsahuje priamo nástroj alebo funkcie určené na detekciu objektov v obrázku.

Optimalizácie

Knižnica je sama o sebe dobre optimalizovaná, o čom svedčí i použitie jej odnože v knižnici *Eblearn*. Nie je to však jediná optimalizácia. Používa knižnicu pre podporu viac procesorov *OpenMP* a podporuje prenesenie výpočtu na GPU pomocou programovacieho jazyku *CUDA*, ktorý podporujú grafické karty od spolocnosti NVidia. Na testovanie rýchlosti použijeme iba optimalizáciu pomocou *OpenMP*, pretože sme nemali vhodnú grafickú Nvidia kartu. Podľa autorov knižnice v [3] je však zrýchlenie výpočtu počas trénovania siete niekoľkonásobné.

Zhrnutie

Knižnica *Torch 7* je veľmi silný nástroj pre vytváranie a učenie neurónových sietí. Používateľ využíva jednoduchý, ale silný jazyk skriptovací Lua, samotná knižnica je efektívne implementovaná v jazyku C. Je možné nechať učenie na knižnicu, alebo si celý proces prispôsobiť podľa svojich požiadaviek. Pomocou rozširujúcich balíčkov je možné pracovať s obrázkami, videom či spracovávať obrázkové dátové sady. Je veľmi dobre dokumentovaná. Prípadné funkcie z experimentálneho balíčku *nnx* síce dokumentované nie sú, ale sú použité v komentovaných príkladoch. Rovnako kladne hodnotíme komunikáciu autorov pri riešení problémov, ktoré sa vyskytli v začiatkoch používania tejto knižnice.

8.3 Porovnanie a výber

Obe knižnice umožňujú prácu (vytváranie, trénovanie a testovanie) s neurónovými sieťami. *Eblearn* pomocou konfiguračných súborov, *Torch 7* pomocou skriptovacieho jazyka. Príprava a spracovanie vstupných dát, ktoré už potom nemienime meniť, je v oboch knižniciach rovnocenná. V prípade *Eblearn* ak chceme pripravené dáta ďalej použiť, musíme ich uložiť do súboru. V *Torch 7* ich stačí spracovať v skripte a mať ich uložené iba v pamäti. V prípade učenia a testovania nám skriptovací jazyk poskytuje väčšiu voľnosť pri riešení úloh, napríklad v modifikovaní výstupov jednotlivých funkcií. Z hľadiska dokumentácie je na tom momentálne lepšie knižnica Torch, i keď autori *Eblearn* začali dokumentáciu postupne rozširovať. Na základe týchto poznatkov sme si vybrali knižnicu *Torch 7*. Pozrieme sa však ešte na rýchlosť jednotlivých knižníc.

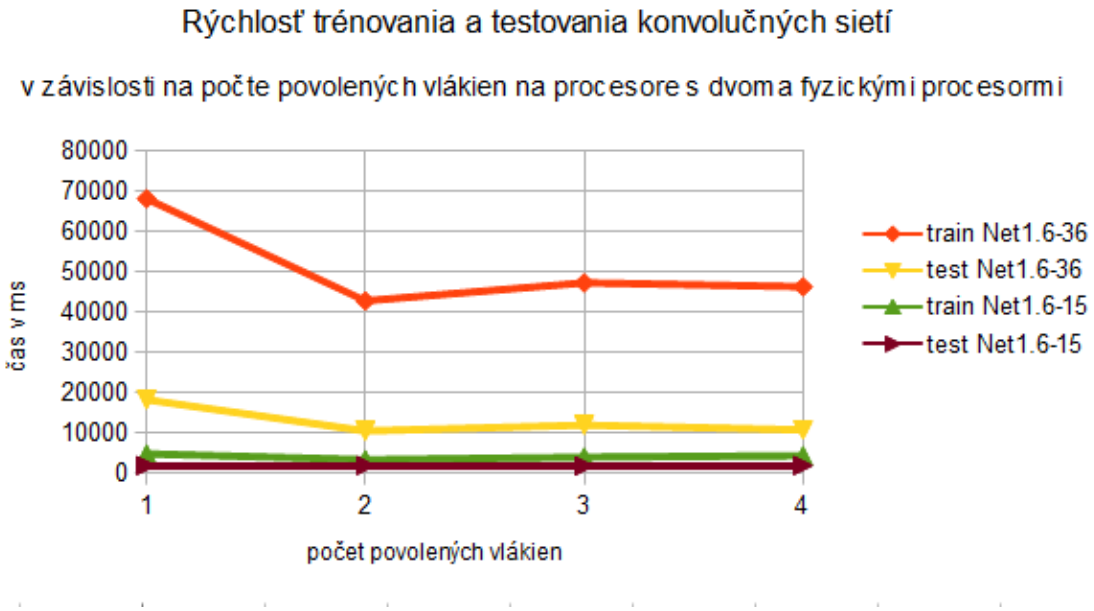
Rýchlosť implementácie knižníc sme chceli otestovať s vyššie popísanými optimalizáciami. Teda knižnicu *Eblearn* s TH a OpenMP a knižnicu *Torch 7* s OpenMP. Bohužiaľ ako sme už spomenuli optimalizácia pomocou OpenMP je v *Eblearn* experimentálna. To sa nám potvrdilo pri testoch, keď program skompilovaný s touto knižnicou nepracoval a skončil chybou. Pre nezaujaté porovnávanie budeme teda porovnávať neoptimalizované knižnice.

Ako testovaciu úlohu sme zvolili rýchlosť naučenia a otestovania 4686 vzorov. Testovali sme na sieti *Net1.6-36* a sieti *Net1.6-15* popísanej v kapitole 5. Test spočíval v načítaní trénovacích obrázkov (tento čas sa do výsledku nezapočítal) a predloženiu každého obrázku sieti, ktorá ho vyhodnotila. Priemerné výsledky z 5 behov pokusu vidíme v tabuľke 8.1. Pozorujeme, že rýchlosť trénovania na veľkej sieti je až približne 1,2krát vyššia a rýchlosť testovania až 2,4 krát vyššia v prospech knižnice *Torch 7*.

Keďže knižnica *Torch 7* plne podporuje výpočet pomocou viacerých procesorov pomocou knižnice OpenMP, pozrime sa ako sa bude meniť rýchlosť v závislosti na počte procesov, ktoré výpočtu povolíme. Testovali sme na procesore Intel® Core™ i5-2450M, ktorý má dve fyzické jadrá. Výsledky vidíme na obrázku 8.1. Vidno, že táto

	Net1.6-36	*Net1.6-15*
Eblearn čas trénovania	87s	7s
Eblearn čas testovania	43s	3s
Torch 7 čas trénovania	68s	5s
Torch 7 čas testovania	18s	2s

Tabuľka 8.1: Rýchlosť trénovania a testovania jednotlivých implementácii.



Obr. 8.1: Rýchlosť knižnice *Torch 7* v závislosti na počte povolených vlákien pri optimalizácii pomocou OpenMP. Test spočíval v testovaní a trénovaní cca 5000 vzorov. Použitý počítač mal iba dva fyzické procesory a vidno, že zvyšovanie počtu vlákien nad počet fyzických jadier, už zlepšenia neprináša, pretože na jednom procesore bežalo vlákien viac a inštrukcie už nemohli byť spracované paralelne.

optimalizácia dosť pomohla. Teda aspoň pri počte vlákien rovných počte fyzických jadier – dvoch. Pri zvyšovaní počtu vlákien, nad počet fyzických jadier, už zlepšenia nenastávali, pretože na jednom procesore bežalo vlákien viac a jednotlivé inštrukcie už nemohli byť spracované paralelne.

Obe knižnice na prácu s neurónovými sieťami sa javia ako vhodné. Optimalizačne a užívateľsky prívetivejšie je na tom knižnica *Torch 7*, nehovoriac o momentálne lepšej dokumentácii a preto sme si túto knižnicu vybrali pre implementáciu nášho riešenia.

Kapitola 9

Implementácia

V kapitole 8 sme vybrali pre implementáciu programu pracujúceho s konvolučnými sieťami knižnicu *Torch 7*. Knižnica je implementovaná v jazyku C, avšak používateľovi – programátorovi poskytuje rozhranie pomocou funkcií implementovaných v skriptovacom jazyku *Lua*.

Pomocou tohto jazyka a knižnice sme implementovali dva programy. Prvý z nich dokáže naučiť konvolučnú neurónovú sieť a druhý, na vstupnom obrázku alebo videu pomocou už naučenej neurónovej siete označí hokejových hráčov.

9.1 Program Train

Náš program *Train* slúži na natrénovanie, prípadne otestovanie, konvolučnej, alebo inej danej neurónovej siete. Trénovanie je na označených obrázkoch rozdelených do kategórii. Program je rozdelený do nasledujúcich častí:

- príprava a spracovanie vstupných vzoriek
- vytvorenie siete
- trénovanie siete
- testovanie siete
- zobrazenie výsledkov

Príprava a spracovanie vstupných vzoriek

Na prípravu a spracovanie vstupných obrazových dát slúži v knižnici *Torch 7* objekt *Dataset* z balíčka *nnx*. Tento objekt dokáže načítať obrázky z danej cesty, zmeniť

ich veľkosť a previesť do štruktúry, s ktorou pracujú ďalšie objekty v tejto knižnici. Pretože načítavanie a zmena veľkosti obrázkov je náročná funkcia, umožňuje tento objekt uložiť výsledný súbor dát do súboru. Príklad volania tohto objektu je nasledujúci:

```
local dataSet = nn.Dataset{
   dataSetFolder="vzory/hraci/",
   cacheFile = "hraci-cache",
   sampleSize{3, 28, 15}
}
```

Tento kus kódu načíta a zmení veľkosť obrázkom z priečinku "vzory/hraci/", výsledný zoznam uloží do súboru "hraci-cache".

My sme pracovali so vzorkami (výrezmi hráčov a pozadí) v rôznych priečinkoch a potrebovali sme pracovať s rôznymi veľkosťami pre rôzne konvolučné siete, preto sme si napísali vlastnú pomocnú triedu *MyDataSet.lua*, ktorá funkčnosť tejto triedy rozširuje o prácu s viacerými množinami dát v rôznych adresároch. Mená pre cache súbory sú taktiež generované automaticky podľa parametrov. Volanie našej triedy vyzerá nasledujúco:

```
local dataSet = nn.MyDataset{
   dataSets={
      {
         name = 'hraci',
         folders = {
            basepath.."samples-cze-rus/test/normalized/0/",
            basepath.."samples-cze-rus/train/normalized/0/",
            basepath.."samples-swe-fin/normalized/0/",
            basepath.."samples-swe-cze/normalized/0/",
         },
      },
      {
         name = 'pozadia',
         folders = {
            basepath.."samples-others/negative/",
            basepath.."samples-cze-rus/train/negative/",
      },
      }
   },
   channels=3,
   sampleSize={ds_width, ds_height},
   cacheFolder = "./cache/"
}
```

Toto volanie spracuje obrázky v zadaných priečinkoch. Je možné zadať viac kategórii - tried, ktoré budeme rozpoznávať. Spracované obrázky uloží pre rýchlejšie načítavanie do cache súborov. Cache súbory budú uložené pre každý spracovaný priečinok. Ich názov sa automaticky vygeneruje zo zadanej cesty a veľkosti výstupného obrázku v našom prípade to budú *vzory-cze-rus-hraci-color-28-15, vzory-cze-swe-hraci-color-28-15, ...*. Táto trieda výrazne uľahčuje prácu s obrázkami v rôznych adresároch.

Vytvorenie siete

Pre prácu s neurónovými sieťami sú v knižnici *Torch 7* pripravené funkcie a objekty z balíčkov *nn*, *nnx* a *optim*. Umožňujú vytvoriť takmer ľubovoľnú neurónovú sieť. Na prácu so sieťami sme vytvorili niekoľko súborov, kde každý súbor obsahoval funkciu pre vytvorenie danej neurónovej siete, napríklad pre sieť *Net1.6-15* vyzerala funkcia takto:

```
function get_network(ds_ch, classes_count)
   local w, h = 15, 24  -- nastavenie veľkosti vstupu siete
   local net_name = "1.6ratio" .. w .. "x" .. h .. "x" ..
      ds_ch .. "classes-" .. classes_count
   -- nastavenie mena siete, pre možnosť sieť
   -- uložiť a jednoznačne ju identifikovať

   local model = nn.Sequential() -- sieť bude vrstevnatá

   model:add(nn.SpatialConvolution(ds_ch, 8, 4, 4))
   -- prvá konvolucna vrstva, vo vrstve ich bude 8
   -- konvolucne jadro ma veľkosť 4x4
   model:add(nn.Tanh()) -- prenosová funkcia
   model:add(nn.SpatialSubSampling(8, 3, 3, 3, 3))
   -- prvá subsamplingova vrstva jadro je veľké 3x4

   model:add(nn.SpatialConvolution(8, 128, 3, 3))
   -- druha konvolucna vrstva
   model:add(nn.Tanh()) -- prenosová funkcia
   model:add(nn.SpatialSubSampling(128, 2, 5, 2, 5))
   -- subsamplingova vrstva
   model:add(nn.Reshape(128))
   model:add(nn.Linear(50))
   -- vrstva klasických neurónov
   model:add(nn.Tanh()) -- prenosová funkcia
   model:add(nn.Linear(50))
   -- vystupne neurony
```

```
   model:add(nn.SoftMax())
   -- normalizovanie výstupu

   return net_name, w, h, model
end
```

Funkcia nám vráti názov siete, ktorý bude slúžiť ako identifikátor, veľkosť vstupu pre sieť a samotnú konvolučnú sieť.

Trénovanie siete

Pre trénovanie siete môžeme použiť priamo funkciu *train*, ale výhodnejšie je rozložiť toto volanie na jednotlivé časti:

```
for i=1,dataSet:size() do
   local sample=dataSet[i]
   -- sample je dvojzlozkove pole, kde prva
   -- zlozka obsahuje vzor, a triedu do ktorej vzor patri
   local input=sample[1]
   local target=sample[2]

   local output=mlp:forward(input)
   -- zratame vystup siete

   criterion= nn.MSECriterion()
   local err=criterion:forward(output,target)
   local gradCriterion = criterion:backward(output,target);
   mlp:zeroGradParameters();
   mlp:backward(input, gradCriterion);
   -- spocitame gradient
   mlp:updateParameters(0.01);
   -- a aktualizujeme vahy
end
-- ďalšie operácie na konci iterácie
-- ...
```

Do cyklu sme ešte pridali poznamenanie si zle vyhodnotených vzorov. Tento zoznam bude slúžiť na zobrazenie vzorov z každej triedy, ktoré boli zle vyhodnotené. Pre ukladanie úspešnosti trénovanej siete, pamätanie si zle vyhodnotených vzorov a kreslenie grafu priebehu učenia sme vytvorili triedu *MyStats.lua*, ktorá tieto činnosti obhospodaruje. Má zákadné metódy:

- *MyStats:add(input, output, target)* - pridanie vzoru, odpovede siete a skutočného požadovaného výsledku do triedy.

- *MyStats:displayNotOk()* - vykreslí vzory z jednotlivých tried, ktoré boli sieťou zle detekované.
- *MyStats:plot()* - vykreslí graf s úspešnosťou sieťe.
- *MyStats:__tostring__()* - vypíše tabuľku s úspešnostami pre jednotlivé triedy a celkovú úspešnosť.

Po každom cykle uložíme celú naučenú sieť a hodnoty úspešnosti siete pre jednotlivé triedy do súboru.

Testovanie siete

Pri testovaní siete prejdeme všetky vzory z danej množiny a necháme sieť spočítať výstup. V cykle si počítame úspešnosť, zapisujeme ju do súboru a rovnako si poznamenávame zle vyhodnotené vzory. Funkcia pre testovanie má parameter, ktorý umožní vzory pred ich otestovaním zašumieť.

Zobrazenie výsledkov

Na konci každej iterácie učenia vypíše program na obrazovku úspešnosť pre jednotlivé kategórie a celkovú úspešnosť. Na konci učenia vykreslíme pomocou balíčka *image* vzory, ktoré boli zle rozpoznané a pomocou metód z triedy *gnuplot* graf zobrazujúci priebeh učenia a testovania.

9.2 Program Detekt

Náš ďalší program *Detekt* slúži na označenie hokejistov na danom vstupnom obrázku alebo videu.

Grafické prostredie programu detekt je vytvorené pomocou balíčkov *qt* a *qtwidget*. Zobrazuje aktuálne spracovaný obrázok. Na začiatku program načíta danú neurónovú sieť a jej parametre, teda veľkosť vstupu a či sa jedná o vstup farebný alebo čiernobiely. Zo siete rovnako zistí počet neurónov v poslednej vrstve. Tento počet určuje počet tried, ktoré je schopná sieť rozpoznávať.

Po načítaní siete načíta buď vstupný obrázok pomocou balíčka *image*, alebo vstupné video pomocou balíčka *ffmpeg*. Pri načítaní videa je možné určiť veľkosť výstupného obrázka, počet snímok za sekundu a dĺžku spracovávaného videa. Pokiaľ ide o video, predkladáme jednotlivé snímky detektoru ďalej, v prípade obrázka sa spracuje iba tento obrázok.

V programe sme vytvorili pole, ktoré obsahuje šírky vzorov, ktoré budeme zo vstupného obrázku vyrezávať. Výška sa vypočíta podľa pomeru strán veľkosti vzoru, ktorý ma na vstupe daná konvolučná neurónová sieť. Napríklad konvolučná neurónová sieť *Net1.6-15-4-32* spracováva obrázok o veľkosti 15x24 px a chceme na vstupnom obrázku testovať výrezy so šírkami 50 px, 70 px a 90 px, budú z obrázku vyrezávané obdĺžniky s rozmermi 50×80 px, 70×112 px a 90×144 px, tie sa zmenšia na veľkosť 15x24 px a predložia sieti na vyhodnotenie.

Pretože sú konvolučné siete odolné voči posunom vo vstupných vzoroch, ako môžeme vidieť na obrázku 6.12 v kapitole 6, nemusíme otestovať všetky výrezy v obrázku. Je možné aby sa i prekrývali. Po niekoľkých pokusoch sme stanovli prekrytie jednotlivých výrezov na 50% z veľkosti výrezu. Menšie hodnoty generovali zbytočne veľa detekcií toho istého hráča na rôznych pozíciách výrezu. Pretože naučenej sieti nevadí ani zmena mierky, je možné prejsť vstupný obrázok iba jednou veľkosťou výrezu. Tu predpokladáme, že klasický záber na hraciu plochu z boku. Šírku tohto výrezu sme stanovli na 10 % šírky vstupného obrazu. Teda pri šírke vstupného obrazu 720 px bude šírka detekčného výrezu veľká 72 px a jeho výška bude 115 px. Prekryv jednotlivých výrezov v ose X bude 36 px a v ose Y 85 px. Touto optimalizáciou sa nám podarilo zmenšiť počet skúmankých vzorov a tak zrýchliť proces detekcie hráčov.

Počet vzorov, ktoré musí sieť otestovať, závisí na veľkosti vstupného obrázku, veľkosti výrezu a prekryve. Počet výrezov je definovaný ako:

$$pocet\_vyrezov\_x = (sirkaobrazku - sirkavyrezu)/velkostprekryvuvosex + 1 \quad (9.1)$$

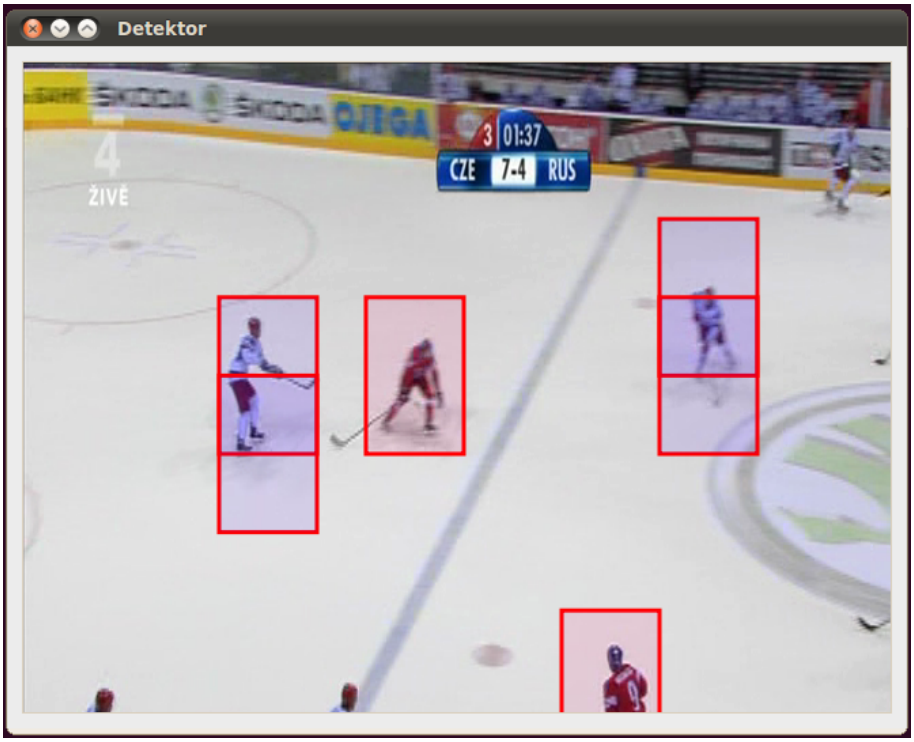
$$pocet\_vyrezov\_y = (vyskaobrazku - vyskavyrezu)/velkostprekryvuvosey + 1 \quad (9.2)$$

$$celkovy\_pocet = pocet\_vyrezov\_x * pocet\_vyrezov\_y \quad (9.3)$$

Teda pre veľkosť obrazu 720×576 px a veľkosť výrezu 72×115 px, je potrebné otestovať 122 vzorov.

V programe cyklom prechádzame jednotlivé výrezy a ktoré zmenšíme do rozmerov požadovaných sieťou. Pretože má výrez i vstup siete rovnaký pomer strán, nedochádza k deformáciám. Zmenšený alebo zväčšený obrázok predložíme sieti a necháme ho vyhodnotiť. Sieť nám vráti vektor, kde každá jeho zložka obsahuje pravdepodobnosť danej triedy (zaisťuje modul SoftMax na konci siete). Vezmeme triedu s najväčšou pravdepodobnosťou a túto pravdepodobnosť porovnáme s prahom, ak je pravdepodobnosť nižšia ako prah, vyhodnotenie zamietneme. Pre našu natrénovanú sieť sa osvedčil prah 0,5. Menší generoval detekcie i na miestach, kde hráč nebol a vyšší nezachytil niektorých hráčov.

Z detekovaného hráča vypočítame jeho priemernú farbu a vložíme ju do zoznamu pre zhlukovací algoritmus *K-means*, ktorý sme v jazyku Lua impementovali v triede *MyKmeans.lua*. Jednotlivé farby hráčov si pamätáme i zo snímok, ktoré sme už spracovali až do maximálneho poču, v našom prípade 150.



Obr. 9.1: Zobrazenie výstupu programu *detect-hokej.lua*.

Nakonci do snímku zakreslíme obdĺžniky na miestach, kde sieť označila, že sa tam nachádza hokejový hráč. Obrázok 9.1 zachycuje výstup našeho programu.

Vytvorené programy sú riadne dokomentované a tak v prípade záujmu o ďalšie podrobnosti, odporúčame nahliadnuť priamo do nich.

9.3 Program Označ hráčov

Počas písania tejto práce sme vytvorli ešte pomocný program na prípravu vzorov na testovanie a trénovanie siete. Je napísaný v jazyku C# a jeho funkčnosť je podrobne popísaná v používateľskej dokumentácii v dodatku B.

Kapitola 10

Zhodnotenie a záver

10.1 Záver

V tejto práci sme si naštudovali konvolučné neurónové siete. Tieto moderné modely neurónových sietí umožňujú spracovávať priamo obrazové dáta, preto sa hodia pre detekčné úlohy. Toto tvrdenie sme v tejto práci overili na úlohe detekovania hokejových hráčov na snímkach z hry. Vytvorili sme pomocný program *Označ hráčov* pre prípravu dát označovanie jednotlivých hráčov a ich následné vyrezanie za účelom ďalšieho spracovania pri trénovaní neurónovej siete. Ručne sme označili niekoľko tisíc hráčov. Tieto výrezy slúžili pre natrénovanie siete. Po analyzovaní veľkosti a tvaru týchto výrezov sme navrhli niekoľko konvolučných sietí, tie sme dôkladne otestovali a vybrali najlepšiu sieť. Výber spočíval vo výbere úspešnej a zároveň rýchlej siete. Ukázalo sa, že pre detekciu hokejistov stačí relatívne malá sieť so vstupom o veľkosti 15×24 px. Pre učenie siete sme vytvorlil program *Train* a sieť sme naučili na pripravených vzoroch. Chyba naučenej siete na testovacej množine bola 1,05 % a sieť si poradila i so zašumenými vzormi. Detekovanie objektov, v našom prípade hráčov, v rôznych pozíciach, situáciách a farbách je veľmi náročná úloha a preto považjeme naše výsledky za úspešné. Naučenú sieť sme podrobne analyzovali. Pre implementovanie detektoru sme si vybrali knižnicu *Torch 7*, ktorá umožňuje programové vytváranie konvolučných neurónových sietí. Pomocou tejto knižnice sme vytvorili program na detekovanie hokejistov *Detektor*, ktorý na vstupnom obrázku hľadá hokejových hráčov a pomocou naučenej siete úspešne označi väčšinu hokejistov, ktorí sa na predloženej snímke alebo videu nachádzajú.

10.2 Ďalšie smery a možnosti rozvoja

V práci sme implementovali funkčný detektor objektov pomocou konvolučných sietí, stále však vidíme priestor pre ďalšie zlepšenia. Úspešnosť detekcie klesla pri vyhodnocovaní snímok zo zápasov, z ktorých nepochádzazali trénovacie dáta. Chybu detekcie

by sme mohli zmenšiť zväčšením trénovacej množiny. Množina v tejto práci obsahuje cca 3500 výrezov hráčov a rovnaký počet automaticky vygenerovaných pozadí. Variánt farieb dresov hráčov je však veľa, preto by bolo dobré túto množinu rozšíriť, aby obsahovala vzorky z čo najviac zápasov. Potom by sieť mala menšie problémy s detekciou hráčov v nových zápasoch. Bohužiaľ sa tým predĺži čas naučenia siete. Ďalšiu možnosť zlepšenia je kombinácia metód pre sledovanie objektov. Nemuseli by sme detekciu vykonávať každú pre každú snímku a vyhľadanie hráča pomocou konvolučnej siete by prebiehalo iba raz za čas.

Dodatok A

Obsah priloženého DVD

Priložené DVD obsahuje následujúcu aresárovú štruktúru:

- *koreňový adresár*
  - *Samples* - výrezy hráčov a pozadí pre učenie a trénovanie sietí
  - *Programs* - vytvorené programy
    - *OznacHracovCS* - adresár s projektom pre MS Visual studio s programom na prípravu a vyrezávanie vzorov
    - *Hokej* - adresár obsahuhúci program na trénovanie a testovanie siete a program na detekiu hráčov
      - ... - potrebné súbory a adresáre
      - *detect-hokej.lua* - program na detekciu hokejových hráčov
      - *train-hokej.lua* - program na učenie a trénovanie konvolučnej siete
  - *diplomovapraca.pdf* - elektronická verzia tejto práce

Dodatok B

Používateľská dokumentácia

B.1 Program Označ hráčov

V tejto podkapitole si popíšeme program, ktorý sme naprogramovali pre prípravu dát na trénovanie a testovanie neurónovej siete *Označ hráčov*. Používateľ dokáže pomocou tohto programu na vstupných obrázkoch označiť chcené objekty, ktoré potom program vyreže a uloží ich pre ďalšie použitie. Ak je nutné, uloží výrezy i s požadovaným pomerom strán. Môžeme označiť objekty až z troch kategórii, plus naviac nám program vyrezáva pozadia. Program k spusteniu potrebuje systém Windows s frameworkom .Net[1].

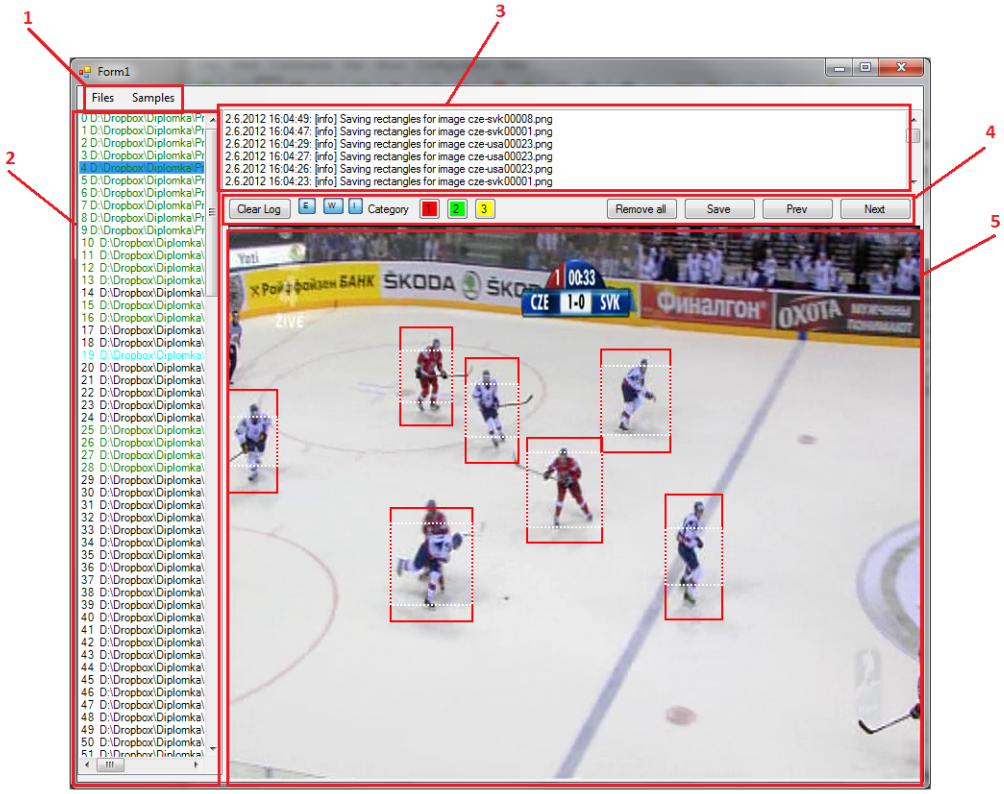
Na obrázku B.1 vidíme hlavné okno programu. Skladá sa z niekoľkých častí:

- Menu
- Zoznam súborov
- Konzola
- Nástrojová lišta
- Obrázok a výrezy

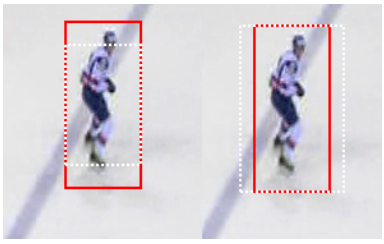
Menu

Menu programu obsahuje dve položky. Prvá z nich je *Files*, obsahuje dve podpoložky: *Open* a *Exit*. Položka *Exit* program ukončí. Pri výbere položky *Open* sa otvorí dialógové okno pre načítanie vstupných súborov. Môžeme otvoriť obrázky s príponami

[1] Testované na systéme Windows 7



Obr. B.1: Program pre prípravu vzorov. Skladá sa z piatich častí. 1 - Menu, 2 - Zoznam súborov, 3 - Konzola, 4 - Nástrojová lišta, 5 - Obrázok a výrezy



Obr. B.2: Znázornenie vpísaného a opísaného obdĺžniku s daným pomerom strán. Červený obdĺžnik označil používateľ. Normalizované obdĺžniky sú označené bielou prerušovanou čiarou. Naľavo vidíme vpísaný normalizovaný obdĺžnik s pomerom strán (šírka k výške 1:1,6), napravo jeho opísaná varianta.

jpg, jpeg a png. Je možné vybrať viac obrázkov naraz. Po výbere sa naplní zoznam súborov názvami otvorených súborov a otvorí sa prvý obrázok.

Položka v menu *Samples* obsahuje nastavenia potrebné pri označovaní a vyrezávaní jednotlivých výrezov. Podpoložky sú nasledujúce:

- *Save samples* - ak je táto položka zaškrtnutá, budú sa ukladať výrezy označené používateľom. Tieto výrezy sa budú ukladať do priečinku *raw-samples/*{*trieda*}, kde trieda je číslo 1, 2 alebo 3 podľa triedy daného výrezu.
- *Save ratio normalized samples* - ak je táto položka zaškrtnutá, budú sa ukladať výrezy označené používateľom avšak upravené podľa zadaného pomeru strán a podľa položky *Inner rectangle*. Tieto výrezy sa budú ukladať do priečinku *normalized/*{*trieda*}, kde trieda je číslo 1, 2 alebo 3 podľa triedy daného výrezu.
- *Save negative samples* - ak je táto položka zaškrtnutá, program vyreže náhodné výrezy z obrázku, ktoré sa nepretínajú s výrezmi označenými používateľom. Tieto výrezy budú mať nastavený pomer strán, ktorý používateľ nastavil pomocou položky *Set ratio*. Uložené budú v priečinku *negative*
- *Inner rectangle* - ak nie je zaškrtnutá, tak výrez s nastaveným pomerom strán je opísaný obdĺžniku vytvorenému používateľom, ak je zaškrtnutá, tak to bude obdĺžnik vpísaný. Príklad môžeme vidieť na obrázku B.2.
- *Delete old samples before saving* - ak zaškrtneme, tak pri uložení sa zmažú výrezy pre daný obrázok. Pokiaľ to zaškrtnuté nie je, výrezy uložené v minulosti sú uchované.
- *Set ratio* - umožňuje nastaviť pomer strán pre uloženie normalizovaných vzoriek. Nastavuje sa reálnym číslom vyjadrujúcim pomer výšky k šírke.
- *Set samples from all images now* - po stlačení tejto položky dôjde k uloženiu výrezov zo všetkých načítaných obrázkov. Rešpektujú sa však pravidlá pre ukladanie popísané vyššie.

Zoznam súborov

Obsahuje zoznam načítaných súborov. Jednotlivé položky môžu mať rôzne farby, ktoré označujú stav jednotlivých obrázkov.

- *čierna farba* znamená, že obrázok ešte nebol spracovaný,
- *zelená farba* znamená, že do obrázku používateľ zaznačil nejaké výrezy,
- *tyrkysová farba* znamená, že užívateľ obrázok otvoril, ale nezaznačil do neho žiadne výrezy.

Konzola

Konzola informuje používateľa o tom, čo program práve robí. Obsahuje tri druhy správ a to *Info - informačné, Warning - výstražné a Error - chybové*. Nové položky pribúdajú do konzoly zhora. *Info* hlášky sú správy o práci programu. *Warning* je chyba, ktorá nie je závažná. Napríklad, že nemôžeme uložiť normalizovaný výrez, pretože zasahuje mimo obrázok. A *Error* sú chybové hlášky, keď sa niečo nepodarí. Konzolu je možné zmazať z nástrojovej lišty pomocou tlačidla *Clear log*. Prípadne si môžeme nechať zobraziť iba jednotlivé typy hlášok, a to tlačidlami *E, W, I*.

Nástrojová lišta

Okrem ovládania konzoly popísanom v predchádzajúcom odseku obsahuje táto lišta tlačidlá na výber kategórie vzoru, a to tlačidlá *1, 2, 3*. Ďalej obsahuje tlačidlo na zmazanie všetkých označených vzorov na práve otvorenom obrázku – tlačidlo *Remove all*. Tlačidlo *Save* na spracovanie otvoreného obrázku a tlačidlá *Prev* a *Next* slúžiace na prechod na predchádzajúci, resp. ďalší obrázok. Pri prechode je aktuálny obrázok automatický spracovaný - teda sú uložené výrezy zadané používateľom. Pokiaľ ďalší obrázok ešte nebol spracovaný, prenesú sa do neho aktuálne používateľom zadané výrezy. To sa hodí najmä pri spracovávaní sekvencie snímok, ktoré na seba nadväzujú, a teda dá sa predpokladať, že rozmiestnenie vzorov bude podobné a obdĺžniky bude postačovať mierne popresúvať.

Obrázok a výrezy

Táto časť programu zobrazí vybraný obrázok. Pokiaľ používateľ mal tento obrázok v minulosti už otvorený, vykreslia sa i výrezy, ktoré boli dávnejšie zadané.

Pomocou ľavého tlačidla myši naťahujeme výrezy, ktoré chceme spracovať. Výrez je reprezentovaný obdĺžnikom, ktorý ma farbu podľa vybranej kategórie. Pomocou ľavého tlačidla myši môžeme vytvorené výrezy posúvať, pomocou pravého tlačidla výrez zmažeme. Okolo výrezu sa bielou prerušovanou čiarou vykreslí ešte jeden obdĺžnik. Je to obdĺžnik, ktorý sa použije pri uložení normalizovaného vzoru. Má zadaný pomer strán a je podľa nastavenia buď vpísaný alebo opísaný danému výrezu.

Výstup programu

Program v priečinku, z ktorého sme načítali obrázky, vytvorí pre každý spracovaný obrázok súbor s názvom *pôvodný-názov-obrázku.rect.txt*, ktorý obsahuje zadané výrezy pre jednotlivé kategórie. Ďalej pri spracovaní obrázkov vytvorí priečinky *negative*, ktorý bude obsahovať výrezy pozadia, *raw-samples* - obsahuje nezmenené

výrezy, tak ako ich označil používateľ a priečinok *normalized* obsahuje normalizované vzorky, teda s daným pomerom strán. Výstupné obrázky sú vo formáte png[2], ktorý je bezstratový, a tak môžeme výstup bez straty kvality ďalej použiť.

B.2 Spustenie programov Train a Detektor

Na rozdiel od programu *Označ hráčov* sú programy *Train* a *Detektor* napísané pomocou knižnice *Torch 7*. Táto knižnica bola v čase písania tejto diplomovej práce spustiteľná iba na systéme linux. Pre spustenie je preto potrebné, aby knižnica *Torch 7* bola správne nainštalovaná v systéme. My sme vyvíjali pod operačným systémom Ubuntu 10.04 64bit. Napíšeme si stručný návod, čo potrebujeme urobiť, aby nám programy išli spustiť.

Najprv je potrebné stiahnuť a nainštalovať knižnicu *Torch 7*. Návod na inštaláciu je na domovských stránkach programu *http://www.torch.ch/*. Náše rogramy boli testované s verziou knižnice k dátumu 30. 7. 2012. Knižnicu ide rozširovať pomocou tzv. balíčkov. Napríklad, pokiaľ chceme požívať funkcie na prácu s obrázkami, je potrebné nainštalovať balíček *image*. Naše programy využívajú viacero ponúkaných balíčkov. Konkrétne potrebujú tieto: *image, nnx, optim, ffmpeg, inline a xlua*. Balíčky sa do knižnice *Torch 7* inštalujú príkazom *torch-pkg install "meno balicku"*. Pokiaľ máme všetko dobre nainštalované, môžeme programy *Train* a *Detektor* spustiť.

B.3 Program Train

Program *Train* slúži na trénovanie neurónovej siete pomocou výrezov s hokejistami. Program je uložený v súbore *train-hokej.lua*. Jeho spustenie je nasledujúce:

```
torch train-hokej.lua <parametre>
```

Výpis prepínačov získame zavolaním príkazu

```
torch train-hokej.lua --help
```

Prepínačmi môžeme meniť správanie programu. Vysvetlíme si ich použitie:

- *-s, --save* určuje, či po trénovaní siete sa jej parametre uložia do súboru. Názov súboru je definovaný názvom a parametrami učenia. Predvolená je hodnota *true.*

[2]Portable Network Graphics

- *-o, --outdir* určuje priečinok, kam sa budú ukladať parametre siete a logy obsahujúce úspešnosť siete počas učenia. Predvolená je hodnota *./results/*.
- *-c, --ch* určuje, či budeme spracovávať čiernobiele obrázky – hodnota 1, alebo farebné obrázky – hodnota 3. Predvolená je hodnota *3*.
- *-cf, --cachefolder* určuje priečinok, kam sa uložia obrázky spracované pred učením/testovaním siete. Priečinok slúži na rýchlejšie znovunačítanie obrázkov. Predvolená je hodnota *./cache/*.
- *-l, --load* cesta k naučenej sieti. Ak nie je zadané, sieť sa inicializuje náhodne.
- *-n, --networktype* definujeme akú sieť ideme učiť alebo testovať. Možné hodnoty sú: *Net1.6-15, Net1.6-36, NetSquare16 a NetSquare36* Tento parameter je povinný.
- *-i, --inner* ak je použitý tento prepínač, budeme sieť učiť na vpísaných štvorcoch, inač na opísaných. Platí pre typ siete *NetSquare16 a NetSquare36*
- *-d, --dataset* definuje priečinok, v ktorom sú uložené výrezy hráčov. Predvolená je hodnota *../../Samples/*.
- *-r, --testratio* určuje, koľko vzorov bude použitých na testovanie. Predvolená hodnota je *0,1*, čo znamená, že na testovanie bude použitá desatina načítaných vzorov.
- *-v, --visualize* ak je použitý tento prepínač, bude sieť zobrazovať priebeh učenia graficky.
- *-ve, --visepoch* určuje po koľkých iteráciách (epochách) učenia sa zobrazí graf priebehu. Predvolená hodnota je *50*.
- *-vn, --visnotok* ak je použitý tento prepínač, okrem grafu priebehu učenia sa zobrazia i vzory, ktoré boli sieťou zle klasifikované.
- *-rs, --seed* číslo na fixovanie generátora náhodných čísiel.
- *-me, --maxepoch* maximálny počet iterácii pri trénovaní.
- *-p1, -p2 a -p3* dodatočné parametre siete. Použité sú iba parametre *-p1 a -p2* a to pri type siete *Net1.6-15*. Určujú počet príznakových máp v prvej, resp. v druhej konvolučnej vrstve.
- *-tn, --testname* názov testu. Ak je zadaný pripíše sa tento názov k názvom súborov obsahujúcich parametre siete a logy priebehu učenia.
- *-nt, --notrain* ak je zapnutý, nebude sa sieť trénovať.
- *-nte, --notest* ak je zapnutý, nebude sa sieť testovať.

- *-ntee, --notestextra* ak je zapnutý, nebudú sa testovať hokejisti z rôznych zápasov.
- *-tns, --testnoised* ak je zapnutý, otestuje sa i zašumená testovacia množina.
- *-np, --noiseparam* veľkosť šumu. Predvolená hodnota je *0,1*.

Pokiaľ by sme chceli natrénovať sieť *Net1.6-15* s tým, že by sme chceli testovať i zašumené vzory, použili by sme príkaz:

```
torch train-hokej.lua -n Net1.6-15 --testnoised
```

Počas behu program vypisuje priebeh trénovania a testovania do konzolového okna. Je možné vidieť úspešnosť na jednotlivých množinách a časy testovania/trénovania.

B.4 Program Detektor

Program *Detektor* umožňuje pomocou natrénovaných sietí označiť na vstupnom videu, alebo obrázku hokejistov. Ovláda sa, podobne ako program *Train*, pomocou príkazového riadka. Po spustení sa zobrazí okno s obrázkom alebo videom, kde zaznačí detekovaných hokejistov.

Program sa ovláda pomocou týchto prepínačov:

- *-n, --networktype* typ siete. Napríklad *Net1.6-35*. Predvolená je hodnota *Net1.6-15*.
- *-l, --load* cesta k súboru s naučenou sieťou, ktorá rozpoznáva hokejistov od pozadia. Predvolená je hodnota *./trained-nets/1.6ratio15x24x3classes-2-p1-4-p2-32.net.ascii*.
- *-i, --image* cesta k vstupnému obrázku.
- *-v, --video* cesta k vstupnému videu.
- *-l2, --load2* cesta k súboru s naučenou sieťou, ktorá rozpoznáva hokejistov od rozhodcov (experimentálna voľba). Predvolená je hodnota *./trained-nets/1.6ratio15x24x3classes-2-tn-hrac-rozhodca.net.ascii*.
- *-k, --kmeans* určuje, či sa pokúsime odhadnúť, do ktorého tímu patrí hráč. Tento odhad robíme pomocou algoritmu k-means.
- *-ks, --kmeanssize* určuje maximálny počet vzorov, z ktorých budeme odhadovať tím hráča. Predvolená je hodnota *150*.

- *-t, --treshold* určuje minimálny výstup siete, aby bol výrez označený za hráča. Predvolená hodnota je *05*.
- *-t2, --treshold2* určuje minimálny výstup siete, aby bol výrez označený za rozhodcu. Predvolená hodnota je *0,5*.
- *--fps* určuje, koľko snímok za sekundu budeme spracovávať zo vstupného videa. Predvolená hodnota je *10*.
- *--length* určuje, koľko sekúnd zo vstupného videa spracujeme. Predvolená hodnota je *20* (s).
- *--length* určuje, od ktorej sekundy budeme vstupné video spracovávať. Predvolená hodnota je *0* (s).
- *-w, --width* určuje, do akej šírky zmenšíme video pred jeho spracovaním. Predvolená hodnota je *360*.
- *-h, --height* určuje, do akej výšky zmenšíme video pred jeho spracovaním. Predvolená hodnota je *288*.

Pre demo s predvolenými parametrami stačí zavolať:

```
torch detect-hokej.lua
```

Ak by sme chceli spracovať video uložené v súbore cc.mp4 v rozlíšení 640×480 px od 10 sekundy, s tým, že chceme detekovať i tím hráča, použili by sme príkaz:

```
torch detect-hokej.lua -v cc.mp4 --seek 10 -k -w 640 -h 480
```

Literatúra

[1] What is Deinterlacing? Facts, solutions, examples. http://www.100fps.com (k 6. 12. 2012).

[2] Bouvrie, J.: Notes on Convolutional Neural Networks. Technická zpráva, MIT CBCL, Nov. 2006.

[3] Collobert, R.; Kavukcuoglu, K.; Farabet, C.: Torch7: A Matlab-like Environment for Machine Learning. In *BigLearn, NIPS Workshop*, 2011.

[4] Fukushima, K.: Neocognitron: A self-or anizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*, ročník 36, č. 4, 1980: s. 193–202.

[5] Fukushima, K.: Neocognitron: A hierarchical neural network capable of visual pattern recognition. *Neural Network*, ročník 1, č. 2, 1988: s. 119–130.

[6] Fukushima, K.: Neocognitron for handwritten digit recognition. *Neurocomputing*, ročník 51, Apr 2003: s. 161–180.

[7] Fukushima, K.: Neocognitron capable of incremental learning. *Neural Networks*, ročník 17, 2004: s. 37–46.

[8] Fukushima, K.; Miyake, S.: Neocognitron: Anew algorithm for pattern recognition tolerant of deformations and shifts in position. *Pattern Recognition*, ročník 15, č. 6, 1982: s. 455–469.

[9] Hinton, G. E.; Osindero, S.; Teh, Y. W.: A Fast Learning Algorithm for Deep Belief Nets. *Neural Computation*, ročník 18, 2006: s. 1527–1554.

[10] Huhel, D. H.; Wiesel, T. N.: Receptive fields, binocular interaction and functional architecure in the cat's visual cortex. *Biological Cybernetics*, ročník 160, č. 1, 1962: s. 106–154.

[11] LeCun, Y.: LeNet-5, convolutional neural networks (k 6. 12. 2012).
URL `http://yann.lecun.com/exdb/lenet/`

[12] LeCun, Y.; Bottou, L.; Bengio, Y.; aj.: Gradient-Based Learning Applied to Document Recognition. *Proc. of the IEEE*, ročník 86, č. 11, Nov 1998: s. 2278–2324.

[13] R., R.: *Neural networks - a systematic introduction*. Berlin: Springer-Verlag, 1996.

[14] Rumelhart, D. E.; Hinton, G. E.; Williams, R. J.: Neurocomputing: foundations of research. In *Learning internal representations by error propagation*, editace J. A. Anderson; E. Rosenfeld, Cambridge, MA, USA: MIT Press, 1988, ISBN 0-262-01097-6, s. 673–695.

[15] Sonka, M.; Hlavac, V.; Boyle, R.: *Image Processing, Analysis, and Machine Vision*. Chapman & Hall, druhé vydání, 1998.

[16] Trojan, S.; kolektiv: *Lékařská fyziologie*. Praha: Grada, 2003.